Imperial College London
Department of Electrical and Electronic Engineering

Hardware-Efficient Data Compression in Wireless Intracortical Brain-Machine Interfaces

Oscar Wiljam Savolainen

Submitted in part fulfilment of the requirements for the degree of
Doctor of Philosophy in Electrical and Electronic Engineering of Imperial College London
December 2022

Declaration of Originality

This thesis represents my own work unless otherwise stated. Any text, figures, tables or other material that is not my own has been appropriately referenced. Any material that I have adapted for this work has been identified as such and where applicable the original source cited.

Oscar Wiljam Savolainen

Copyright Statement

Abstract

Brain-Machine Interfaces (BMI) have emerged as a promising technology for restoring lost motor function in patients with neurological disorders and/or motor impairments, e.g. paraplegia, amputation, stroke, spinal cord injury, amyotrophic lateral sclerosis, etc. The past 2 decades have seen significant advances in BMI performance. This has largely been driven by the invention and uptake of intracortical microelectrode arrays that can isolate the activity of individual neurons. However, the current paradigm involves the use of percutaneous connections, i.e. wires. These wires carry the information from the intracortical array implanted in the brain to outside of the body, where the information is used for neural decoding. These wires carry significant long-term risks ranging from infection, to mechanical injury, to impaired mobility and quality of life for the individual. Therefore, there is a desire to make intracortical BMIs (iBMI) wireless, where the data is communicated out wirelessly, either with the use of electromagnetic or acoustic waves.

Unfortunately, this consumes a significant amount of power, which is dissipated from the implant in the form of heat. Heating tissue can cause irreparable damage, and so there are strict limits on heat flux from implants to cortical tissue. Given the ever-increasing number of channels per implant, the required communication power is now exceeding the acceptable cortical heat transfer limits. This cortical heating issue is hampering widespread clinical use. As such, effective data compression would bring Wireless iBMIs (WI-BMI) into alignment with heat transfer limits, enabling large channel counts and small implant sizes without risking tissue damage via heating.

This thesis addresses the aforementioned communication power problem from a signal processing and data compression perspective, and is composed of two parts. In the first part, we investigate hardware-efficient ways to compress the Multi-Unit Activity (MUA) signal, which is the most common signal in modern iBMIs. In the second and final part, we look at efficient ways to extract and compress the high-bandwidth Entire Spiking Activity signal, which, while underexplored as a signal, has been the subject of significant interest given its ability to outperform the MUA signal in neural decoding. Overall, this thesis introduces hardware-efficient methods of extracting high-performing neural features, and compressing them by an order of magnitude or more beyond the state-of-the-art in ultra-low power ways. This enables many more recording channels to be fit onto intracortical implants, while remaining within cortical heat transfer safety and channel capacity limits.

Acknowledgements

Thank you, above all, to my mother Tuija, whose support has enabled everything I've ever done. Thank you to my sister Salla, whose support, hard work and sense of humour is an inspiration to all of us. And thanks to Gregory, who has become a great part of the family despite his penchant for atrocious puns.

Thank you to my supervisor Timothy Constandinou, who has given invaluable feedback and guidance. His support has extended beyond just technical matters, and has helped me understand the overall field of Brain-Machine Interfaces as well as the ins and outs of academia and research.

I am grateful to my lab mates who have made my stay at Imperial College enjoyable, educational and interesting. Thank you in particular to Zheng Zhang, who it has been a pleasure to work alongside closely for the last 2 years in developing some of the work in this thesis. I also want to thank Iza in particular, who has helped me navigate the complexities of academic administration.

I am thankful to all of my friends, who have made these past 4 years a pleasure. Finally, I am grateful to the country of Mexico, that hosted me for a long period of my PhD during the remote-working era. It was the most eye-opening experience of my life, and I remain convinced that its people are the nicest, hard-working, and underestimated in the world.

Contents

List of Figures

List of Tables

Acronyms

ASIC Application Specific Integrated Circuit

BDP Behavioral Decoding Performance

BMI Brain-Machine Interface

BP Binning Period

BR Bit Rate

BTR Behavioral Temporal Resolution

CR Compression Ratio

DED Delta Event-Driven

EAP Extracellular Action Potential

ECoG Electrocorticography

EED Explicit Event-Driven

EEG Electroencephalogram

ESA Entire Spiking Activity

FF Flip-Flop

FIR Finite Impulse Response

FPGA Field Programmable Gate Array

FR Firing Rate

GED Group Event-Driven

LFP Local Field Potential

LNNT Linear Neural Network Time

LSTM Long-Short Term Memory

LUT Look-Up Table

M1 Primary Motor Cortex

MUA Multi-Unit Activity

NHP Non-Human Primate

PCA Principal Component Analysis

QRNN Quasi-Recurrent Neural Network

RAM Random Access Memory

SCLV Sorted Codeword Length Vector

SH Static Huffman

SUA Single-Unit Activity

WI-BMI Wireless Intracortical Brain-Machine Interface

Z.Z. Zheng Zhang

Chapter 1

Introduction

1.1 Motivation

Millions of people around the world are living with complete or partial paralysis because of neurological disorders such as stroke, Traumatic Spinal Cord Injury, Amyotrophic Lateral Sclerosis, and Parkinson's. Millions more are suffering from a loss of motory ability from amputation. As a whole, movement disorders come with significantly reduced quality of life for the individuals and their loved ones. These can include reduced autonomy and ability to participate in life, reduced employment opportunities and athletic performance, and increased risk for other diseases. Furthermore, the financial burden for public and private healthcare systems, as well as society at large, can be enormous. For example, 12,000 thousand new people experience Traumatic Spinal Cord Injury every year in the United States, with an estimated total of approximately 273,000, with costs for each patient often quickly increasing into the hundreds of thousands of dollars [1]. To improve the lives of the patients and their loved ones, as well as alleviate the financial burden to both individuals and society, effectively treating motor-impairments is of significant importance.

Brain-Machine Interfaces (BMIs) have emerged as a possible solution for treating motor-impairments. By measuring activity in the motory cortices in the brain, damaged parts of the motory nervous system and body can be bypassed, allowing one to control an avatar. According to patient surveys, patients' highest priority in terms of restoring ability is to restore hand kinematics [2–4]. In the past two decades, significant progress has been made towards this goal, with Non-Human Primates (NHPs) and human patients using BMIs to control computer cursors [5–9] and robotic arms [10–13].

The state-of-the-art in BMIs consists of using intracortical microelectrode arrays to measure neural activity in the brain. This has very high spatial and temporal resolution, to the degree that one can distinguish between individual neurons firing in the vicinity of the electrode [14].

However, a major obstacle to widespread clinical application is that intracortical BMIs currently use percutaneous connections, i.e. wires. These wires serve to transfer the data from the intracortical electrodes to an external decoder outside the brain, where computationally intensive processing of the neural data is done. These percutaneous connections come with significant risks in terms of infection and mechanical injury. They also significantly degrade the quality of life of the individual. As such, there has recently been significant interest in exploring Wireless Intracortical Brain-Machine Interfaces (WI-BMIs). In WI-BMIs, the information is wirelessly communicated out from the brain, e.g. with the use of radio waves. However, this consumes significant amounts of power. This is a problem because any power that is used on-implant is dissipated as heat into the neural medium. If too much heat is transferred into cortical tissue, the tissue can be permanently damaged. Therefore there are strict limits on acceptable heat flux amounts into cortical tissue. Furthermore, the communicated wave can also be absorbed by the neural medium, which

can cause further heating, or even injury due to rapid pressure changes in the case of ultrasound. Therefore, communication power must be limited.

Driven by the desire for more channels on-implant, the communication power is exceeding what is acceptable in terms of power use. This is most notable in the case of free-floating mm or sub-mm scale implants, which cannot communicate even a single raw broadband (i.e. full-spectrum) neural recording without exceeding the power budget. As such, some form of data compression of the raw intracortical signal is warranted. This is the topic this thesis will explore.

Data compression comes in two different forms: lossless and lossy. Lossless compression leverages information theory to assign shorter codewords to more likely symbols, compressing the data without losing any information [15]. However, WI-BMIs have limited processing power and memory, and lossless compression algorithms are generally not designed for ultra-low resource environments such as WI-BMIs. As such, any lossless compression techniques that are used should be tailored to the WI-BMI environment.

The second form of compression is lossy compression. This involves transforming the data so as to remove or degrade aspects of it that are assumed to not be of interest to the final application. This concentrates the relevant information in the data into fewer bits, reducing the bandwidth. In the case of motory BMIs, the final application is, as guided by patient interest surveys, typically hand kinematics. Therefore, the objective of lossy compression in motory BMIs is to eliminate as much superfluous information as possible from the neural signal, while taking care to not degrade the decoding ability of the BMI. Furthermore, any on-implant compression should be done using minimal hardware, to reduce the size of the implant. Additionally, the communication power savings should be sufficient to warrant the additional processing power required for the compression.

There are a small number of common, well-understood lossy compressions of the raw broadband intracortical signal. These are typically divided into the Local Field Potential (LFP)-based features, and spike-based features. Most intracortical BMIs target spikes, i.e. neural impulses. This is because they are understood to represent the main unit of information processing in the brain. Spikes are typically encoded in one of three different forms: Single-Unit Activity (SUA), Multi-Unit Activity (MUA), Entire Spiking Activity (ESA). SUA consists of taking the spikes, and sorting them by spike shape, where similarly shaped spikes are assumed to originate from the same putative neuron. As such, SUA gives us the neural firing rates of individual neurons in the electrode vicinity. MUA is similar, but the spikes are not sorted: all spikes on the same electrode are assigned to the same putative neuron. ESA is an analogue representation of MUA. It is obtained by highpass filtering the broadband signal, rectifying the signal, and enveloping the result. As such, it gives an envelope of unsorted spiking activity on a channel. Since LFPs and the mentioned spike-derived features are well-understood lossy compressions of intracortical broadband signals, they represent an excellent starting point for investigating the hardware efficient compression of data in WI-BMIs.

1.2 Research Objectives

This thesis aims to develop hardware-efficient lossless and lossy data compression algorithms for the compression of intracortical neural recordings. In develop these algorithms, where possible, four metrics are tracked.

1. Total on-implant dynamic power, defined as the sum of the communication and processing power on-implant.

2. The required hardware resources for the implementation of the data compression algorithms, i.e. the amount of Flip-Flop (FF)s, Look-Up Table (LUT)s, and memory (Read-Only Memory and Random Access Memory (RAM)).

3. Behavioral Decoding Performance (BDP), which measures the similarity between the 'true' behavior and that given as the decoded behavioral output by the motory BMI. In this thesis, BDP is defined as the mean Pearson correlation coefficient between the BMI-predicted and observed X and Y-axis velocities when decoding hand movements.

4. The fourth is Behavioral Temporal Resolution (BTR), which consists of the temporal resolution of the decoded behavior. This should be sufficiently high to enable a seamless control loop for the user. A typical target for BTR is 30 ms or less, although values as high as 100 ms also occur in the literature.

Together, these four metrics give a holistic understanding of the balance between power, hardware resources, BDP and BTR, for the different data compression algorithms. Our objective is to minimize power and hardware resources, while maximising BDP and giving as fine a BTR as possible. This optimisation problem, along with the typical data flow for WI-BMIs, is shown in Fig. 1.1.

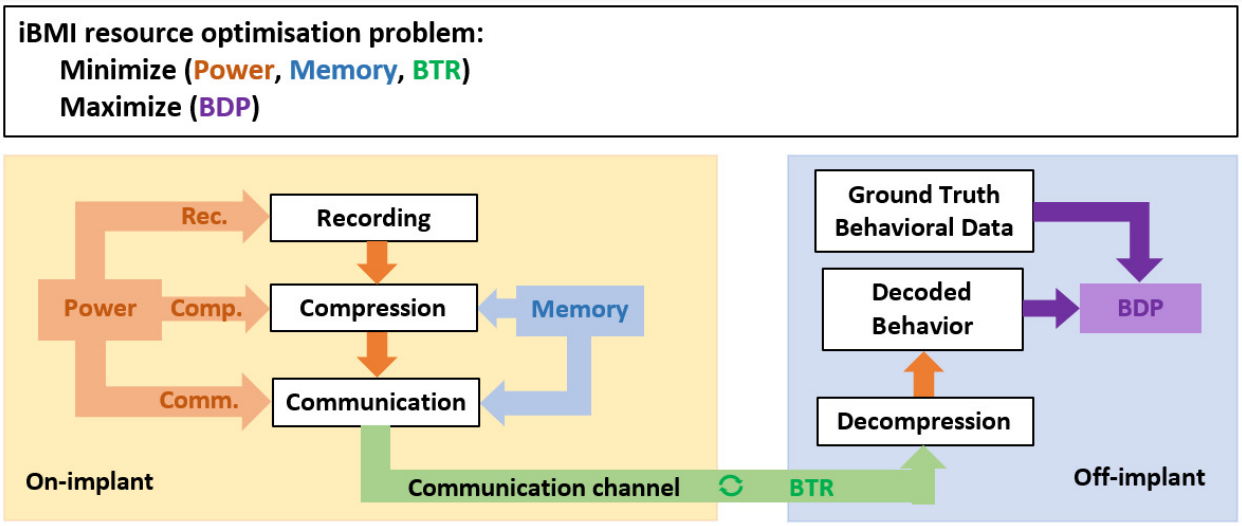


Figure 1.1: Data flow in behavior-decoding BMIs. Also shown is the optimisation problem of on-implant power, resources behavioral decoding performance and temporal resolution.

To achieve this objective, we set two main objectives for this thesis:

- *Compress the MUA signal.* As the MUA signal is the most common signal in modern WI-BMIs, it warrants starting our work there. The standard representation of the MUA signal involves 1-bit sampling at a sampling frequency of 1 kHz, where channels are multiplexed. However, this translates into a Bit Rate (BR) of 1000 bps/channel. For many WI-BMI applications, this is an unacceptably high bandwidth. Therefore, it is necessary to compress the MUA signal to enable a broader range of applications.

- *Compress the ESA signal.* The ESA signal is obtained by highpass-filtering the broadband signal, and then rectifying and enveloping it. It has been shown to have state-of-the-art BDP [16], outperforming even the MUA signal. Considering that it is quite simple to compute from the broadband signal, it seems like a very attractive signal for motory WI-BMIs. However, its BR is unacceptably high, typically sampled at 1 kHz and 16 bits. As such, it is important to investigate whether the ESA signal can be extracted and compressed in a hardware-efficient, low-power way that balances the four metrics.

1.3 Thesis Outline

This thesis is composed of two parts, each tackling one of the aforementioned research objectives. The structure of this thesis is as follows:

- Chapter 2 | Brain-Machine Interfaces: Background and State of the Art

  This chapter gives the necessary background on BMIs, with a focus on signal processing and data compression. We start by giving a general overview on BMIs, including history and different BMI types. We discuss heat limits in BMIs, and give background on data compression in the context of resource-constrained environments such as WI-BMIs. We give an overview of different common intracortical neural features, in the context of WI-BMI data compression. Finally, we give an overview of BMI behavioral decoding, as well as different decoders.

Part I: Compressing Multi-Unit Activity

- Chapter 3 | Static Huffman Compression of Intracortical Neural Signals

  In this chapter, we investigate the use of Static Huffman (SH) encoders for the compression of a broad class of intracortical neural signals. This was a preliminary investigation into whether SH encoders are appropriate or not for intracortical signals, and did not consider the MUA signal. This is because the MUA signal requires more processing than the considered LFP, EAP and ESA signals, and so is dealt with in detail in the next chapter.

- Chapter 4 | Static Huffman Compression of MUA

  In this chapter, we use windowing to lossily compress the MUA signal, and tailor the SH encoder to the resulting signal. We introduce various hardware-efficient techniques for improving SH compression of MUA, including sample histogram mapping and the use of multiple encoders with assignment. We perform a large grid search of the parameter space, and present the results in terms of the four key metrics: total dynamic power, hardware resources, BDP and BTR. Finally, we test a specific system that showed top performance, and present the results.

- Chapter 5 | Comparison to Event-Driven Architectures for Compressing MUA

  We develop various event-driven SH compression schemes for MUA that tackle some of the weaknesses of the windowing method, specifically poor BTR. We introduce these schemes, and analyse their compression performance and hardware complexity. Finally, we give recommendations of which algorithm is best given the implant conditions, such as desired BTR and the number of channels hosted on-implant.

Part II: Compressing Entire Spiking Activity

- Chapter 6 | Minimum Requirements for the Processing and Compression of ESA

  In this chapter, we investigate low-complexity and hardware-efficient ways to extract and compress the ESA signal. We minimize the number of on-implant computations, e.g. by using delta-sampling as a means of highpass filtering, and windowing as a means of enveloping and downsampling. We perform a grid search, and analyse the results in terms of BR, BDP and BTR.

- Chapter 7 | Conclusion and Future Directions

  This chapter concludes the work, and highlights novel contributions. It also highlights potential future research directions.

Chapter 2

Brain-Machine Interfaces: Background and State of the Art

This chapter considers general background on WI-BMIs. It introduces common intracortical neural signals, the problem of thermal dissipation in WI-BMIs, the fundamentals of data compression, as well as how data compression applies to the WI-BMI environment. Finally, we present the fundamentals of BMI behavioral decoding, and the datasets used in this thesis.

This chapter has been adapted from the following published articles:

Savolainen, Oscar W., et al. "Hardware-Efficient Compression of Neural Multi-Unit Activity." bioRxiv (2022).

2.1 History of Brain-Machine Interfaces

BMIs, also referred to as Brain-Computer Interfaces (BCI), are electronic devices that measure neural activity, extract features from that activity, and convert those features into outputs that replace, restore, enhance, supplement, or improve human functions. They allow the individual to interact with their environment, using their thoughts alone, regardless of other neurological or physical impairments. They can, for example, be used to treat paraplegia, quadriplegia, movement disorders, Locked-in syndrome and more [17, 18].

BMI research originated in 1968, when Evarts showed that the Firing Rates (FRs) of individual neurons in the Primary Motor Cortex (M1) of NHPs [19] were correlated with the movements of the wrist. In 1977, Vidal showed that a computer cursor could be moved around a screen with Electroencephalogram (EEG) signals [20].

BMI research accelerated significantly in the 1990's, with the advent of intracortical microelectrode arrays. Intracortical microelectrode arrays allow the simultaneous recording of the extracellular activity of multiple neurons, and are considered state-of-the-art. In 1996, Kennedy and Bakay carried out the first invasive BMI trial in a human patient suffering from locked-in syndrome due to Amyotrophic Lateral Sclerosis. They used a neurotrophic electrode [21] that measured neurological signals, and showed that they could be voluntarily modulated by the user. In 1999, Birbaumer et al. demonstrated a non-invasive EEG device that allowed fully locked-in patients to type out text onto a computer screen [22]. More advanced intracortical BMI systems were then developed in NHPs for the neural decoding of hand kinematics, to control robotic arms [5, 10, 23] and computer cursors [6, 7, 24]. In 2004, the BrainGate project was launched to accelerate BMI research,

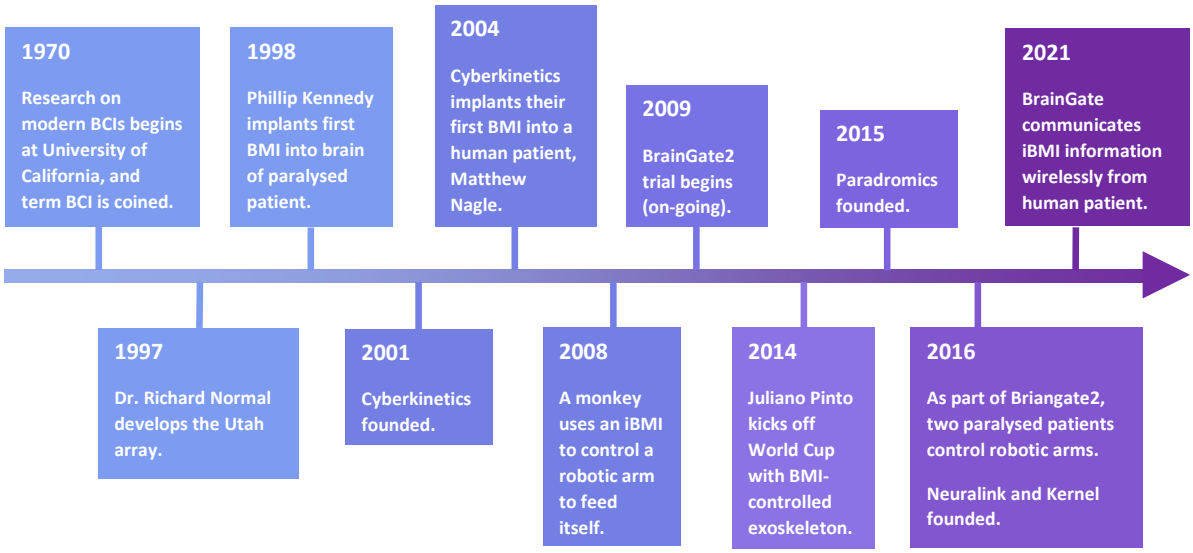


Figure 2.1: Overview of some of the milestones in modern BMI development.

with the aim of translating them into clinical human trials [25]. Multiple milestones were reached, such as that of paralysed patients controlling a computer cursor [11], a robotic arm [26], and even functional electrical stimulation systems [27]. The current state-of-the-art in intracortical BMI decoding was reached when Wodlinger et al. demonstrated a robotic arm with ten degrees of freedom [13].

While BMIs come in different forms, such as EEG, Electrocorticography (ECoG), fMRI, and optical methods, intracortical BMIs are the most invasive form and offer the highest spatial and temporal resolution [14]. intracortical BMIs are capable of measuring the FRs of individual neurons in the vicinity of the electrodes. For example, the Neuropixel probe was reported in 2017 [28]. It consists of multiple multiplexed probe sites micro-fabricated along rigid penetrating silicon shanks, with 384 recording channels that can programmably address 960 sites along the shank. The Neuropixel 2.0, introduced in 2021, has over 5000 sites that can be addressed, also with 384 recording channels [29]. These allow the simultaneous measurement of hundreds of individual neurons, as well as the coverage of different recording locations without invasive relocation of the shank probe.

For an in-depth review of BMIs from basic science to neurorehabilitation, see [30]. For a comprehensive review of intracortical BMIs, see [31]. An overview of some of the milestones in modern BMI development is given in Fig. 2.1. The basic data flow in intracortical BMIs is also given in Fig. 2.2.

2.2 Wireless Intracortical Brain-Machine Interfaces

Intracortical BMIs, as aforementioned, have seen significant development. However, as with the Neuropixel probe, the current intracortical BMI paradigm in clinical trials in humans involves the use of percutaneous connections, i.e. wires, to transfer the measured neural information to outside the body [12, 26, 32]. These percutaneous connections come with significant infection and mechanical health risks, as well as significantly reduced quality of life for the individual [30, 33]. Wired connections are also a significant source of device failure [33], as well as of electromagnetic interference, crosstalk and power inefficiencies [34]. Furthermore, the implantation procedure of wired devices can involve invasively tunnelling for leads, increasing tissue damage and foreign body response [35]. While devices can be made wireless with the use of batteries, these require some form of expert or even surgical intervention to replace, as battery life is finite. Fully wireless

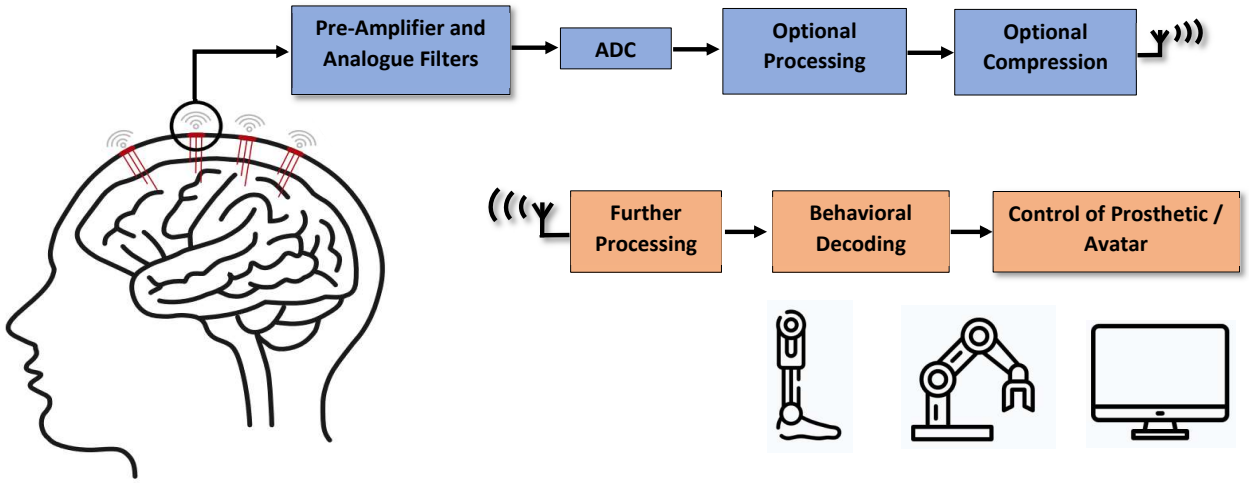


Figure 2.2: Basic block diagram of typical BMI data flow. Optional processing and compression of the recorded data takes place on-implant, and more computationally intensive processing, i.e. advanced feature extraction and/or behavioral decoding, takes place off-implant.

powering would enable chronic use without having to replace the implant. As such, the next generation of intracortical BMIs are desired and expected to be wireless. This is a major remaining bottleneck to widespread clinical use. To overcome this, both the communication and powering must be wireless.

It is also desired that these systems record from different parts of the brain, and that the spatial coverage be both high resolution and customisable. As such, a distributed network of free-floating sub-mm scale implants is particularly attractive [14]. Other than offering attractive spatial coverage, this minimises foreign-body response and tissue trauma due to implantation [35]. Ultra-minimized probe sizes can enable ultra-minimally invasive delivery methods such as laparoscopy or injection [34]. Furthermore, a distributed network increases the robustness of the system, where if an individual probe fails the rest of the system continues to function. This reduces the need for invasive replacement surgery, as well as protecting against the negative effects from the loss of operation of the implant. A distributed sensor system is theoretically promising, however it requires that the individual sensor node size is ultra-miniaturized while maintaining a high-fidelity neural interface [36].

Significant research has gone into the developing communication methods for WI-BMIs. These either use acoustic ultrasound, or electromagnetic radio frequency waves. On the acoustic front, a team at UC Berkeley proposed a system of 10-100 μW scale, free-floating, independent sensor nodes, powered and communicating via ultrasound [37]. However, ultrasound powering can suffer from acoustic link misalignment, where if the piezo-electric crystal's surface is not perpendicular to the incoming wave, the absorbed power is reduced. For example, [34] reported that an *in vivo* mm-scale peripheral nervous system StimDust neuro-simulator ultrasound mote, with ideal link alignment, required an acoustic intensity equal to 7.8% of the safety limit for diagnostic ultrasound of 720 mW/cm. However, with a 75°mote angular misalignment range, the required acoustic intensity increased to 36% of the safety limit. This was for a single mote, although it was a bidirectional peripheral nervous system interface capable of both recording and stimulation.

Various radio drequency powered WI-BMIs have also been proposed. The ENGINI (Empowering Next Generation Implantable Neural Interfaces) platform uses a cranial transponder to inductively couple power to, and communicate data from, a distributed array of mm-scale freely-floating intracortical probes [38]. The cranial transponder then communicates the data to an external decoder. The system targets LFP signals along the cortical column, with 8 intracortical microwire electrodes.

First detailed in 2019, the Neurograin platform uses a distributed system of independent, sub-mm scale nodes, each hosting a single channel [36], each with a 40μW power budget. This was first demonstrated as a distributed network of probes measuring ECoG signals at a 500 Hz bandwidth, but in concept can be applied more broadly. As of the most recent publication, a custom time-division multiple access communication protocol was designed to scale up to 770 Neurograins [39].

Other radio frequency based WI-BMIs, that eliminate the wired connection between a monolithic implant to external electronics, have also been proposed [40–43]. For example, in 2016 Neuralink was founded, which is a private BMI company that seeks to develop and commercialise radio frequency-based WI-BMIs. They use an intracranial transplant that houses a monolithic chip, with feedthroughs for microwire electrodes that are implanted using a 'sewing-machine', Machine Vision robotic surgeon. Powering and communication is achieved via backscattering of radio frequency waves.

2.3 Communication Bandwidth as a Constraint

2.3.1 Heat Limits in intracortical BMIs

Due to heating constraints in cortical tissue, on-implant power usage is strictly limited in WI-BMI to a 1 °C temperature increase or 1.6 mW/g of Specific Absorption Rate in tissue [14, 44, 45]. In the context of heating due to absorption of radio frequency, the IEEE standard C95.1-2019 gives limits for Specific Absorption Rate heating depending on radio frequency frequency [46]. However, it specifies that the understood Specific Absorption Rate limits in brain tissue are generally derived from models and lack rigorous studies in live animals or humans, with significant variance between models [46]. FDA regulations further dictate that the local heat increase of brain tissue due to intracortical implants should be limited to only 0.5 °C [47]. In muscle and lung tissue, it is understood that up to 40 mW/cm^2 heat flux can be allowed, however the limit is likely lower in cortical tissue [44]. For example, a retinal implant device was found to provide a heat flux of 15.5 mW/cm^2 when including the size of the insulation, and the temperature increases were less than a degree [44, 48], which is assumed to be acceptable. In the same study, when the power was increased to represent 62 mW/cm^2, the temperature increases reached as high as 3°C [44, 48], which is unacceptable. While no equivalent study exists on neural tissue, it is best to err on the side of caution. Therefore, throughout this thesis we will assume a maximum heat flux limit of 10 mW/cm^2 to hopefully provide a reasonable safety margin [37], given the extreme paucity of data on the effects of chronic heat flux on cortical tissue [44, 49].

This severely limits the available power on-implant. For example, a 1 × 1 mm scale free-floating intracortical implant, with a maximum permitted heat flux of 10 mW/cm^2, would have a maximum total power budget of 200 μW, assuming equal heat flux from both planes of the implant and negligible heat flux from the edges of the device.

Furthermore, in the case of radio frequency WI-BMIs, any power that is used on-implant needs to be wirelessly transferred to it, which heats the tissue via absorption of radio frequency waves [44]. The current literature shows that, in the best case, only 32% of transmitted power can be recuperated on-implant with an inter-coil distance of 20 mm, meaning that the remaining portion is absorbed by the cortical environment [50]. This value is typically much lower, with efficiencies often lower than 1% [50]. As such, any tissue heating from the power delivery also needs to be accounted for. Every μW that is used on-implant is responsible for another few μWs or even tens of μWs of heating due to transfer loss. Therefore, even a 10 mW/cm^2 heat flux limit may be overly optimistic for WI-BMIs given the effects of wireless powering. As such, methods to reduce on-implant power usage, to minimize all aspects of tissue heating, are highly desirable.

2.3.2 Communication energy estimates

The WI-BMI data communication channels can be described as uplink and downlink. Uplink refers to the data flow from the implant to an external device, and downlink refers to the flow from external device to implant. WI-BMIs generally require a higher uplink data rate than a downlink, since they need to transmit significant amounts of neural data to an external device.

Radio frequency data communication schemes are typically implemented using different types of shift keying (e.g. amplitude, phase, on-off) [51–55]. They are the most power-efficient solutions for radio frequency implants; however, their BR limits are below 20 Mbps. The ultra-wideband uplink proposed in [56] achieved 46 Mbps with 118.3 pJ/bit. The implantable microsystems that are designed and fabricated based on full-custom Application Specific Integrated Circuits (ASICs) are more power-efficient than microcontroller and Field Programmable Gate Array (FPGA) based solutions [57, 58].

Different communication energies that have been achieved in the literature are given in Table 2.1. However, it is worth mentioning that the studies cited in Table 2.1 measured each system's communication energy while outside the neural medium. The neural medium is highly lossy for radio frequency waves [37] with the aforementioned low transfer efficiencies. Therefore, one should assume that the actual communication energy is significantly higher than given in Table 2.1 to account for transfer loss. Furthermore, one should assume that the difference between FPGA and ASIC communication energies is significantly reduced in the neural medium, given that the transfer loss is independent of hardware efficiency and will dominate the processing power differences.

Table 2.1: Wireless radio frequency data communication power consumption comparison. Produced by Peilong Feng in our co-authored work [59].

Publication	Downlink			Uplink		
	Modulation	Bit rate	Power	Modulation	Bit Rate	Power
[51]	ASK	15 kbps	-	Backscatter	-	-
[52]	OOK-PPM	50 kbps	4 nJ/bit	OOK	6.78 Mbps	1.34 nJ/bit
[53]	OOK	1 Mbps	13 pJ/bit	OOK	16 Mbps	50.4 pJ/bit
[54]	ASK	11 Kbps	1.25 nJ/bit	-	-	-
[56]	-	-	-	UWB	46 Mbps	118.3 pJ/bit
[55]	ASK-PWM	1 Mbps	-	BPSK	10 Mbps	-
[60]	-	-	-	Backscatter	0.5 Mbps	240 pJ/bit
[61]	Nordic - nRF24L01+	2 Mbps	16.95 nJ/bit	Nordic - nRF24L01+	2 Mbps	20 nJ/bit
[62]	-	-	-	Backscatter	1.25 Mbps	87 nJ/bit

Throughout this thesis a 20 nJ/bit uplink communication energy [61] is assumed. This is state-of-the-art for FPGAs (Table 2.1). This is an optimistic estimate, however the goal of this thesis is to analyse the effects of data compression. Therefore, if the communication power is higher than 20 nJ/bit, as is likely, then the effect of data compression will be even more significant in terms of power savings. As such, assuming very low communication energy gives a conservative estimate of the power savings obtained from data compression. The motivation for using FPGAs instead of ASICs in this thesis is expanded upon later in Section 2.6.

2.3.3 Communication power estimates

The average communication power per channel can then be calculated from the BR, given in (bps/channel):

$$\text{Comm. power} = \text{BR} \times \text{Comm. energy per bit} \qquad [\text{W/channel}] \quad (2.1)$$

Raw broadband data is typically sampled at 25-30 kHz and 16 bits/sample [17,63–65], although resolutions as low as 7 kHz and 6 bits/sample are possible for applications such as spike detection of large spikes [66]. This creates very large communication bandwidths, ranging from 42-480 kb/s/channel. For 20 nJ/bit, this translates into a communication power demand of 0.84-9.6 mW/channel. For the 1 mm×1 mm scale implant considered earlier, with a power budget of 200 μW derived from the 10 mW/cm^2 heat flux limit, not even a single broadband channel could be communicated off-implant. This is without even considering the power required for recording the signal, or the static power consumption of the implant. As such, some form of data compression is necessary to enable wireless communication.

It also warrants mentioning that similar studies on per bit energy cost have not been undertaken for acoustic ultrasound communication: it is not clear how much power is required to communicate acoustically for WI-BMIs. This is the case for both monolithic and distributed acoustic systems.

A highly optimistic breakdown of WI-BMI power consumption, for an implant of arbitrary size, is given in Fig. 2.3. It shows that, after transfer loss, the power consumption is absolutely dominated by communication power. Furthermore, the transfer loss component is also dominated by the communication power requirement. As such, it makes clear that reducing the communication power is a clear priority. Doing so would reduce the on-implant power, but also the transfer losses.

2.3.4 Channel capacity

Another motivation for on-node data compression is finite channel capacity. There is no consensus on an upper limit for channel capacity in WI-BMIs (Table 2.1). However, the channel capacity is finite given the tissue's finite ability to absorb either acoustic or electromagnetic waves without damage. As such, the BR of each node limits the size of a distributed network of free-floating WI-BMI nodes. For example, in the case of distributed nodes communicating via a time-division multiple access protocol, the amount of available nodes is limited by the size of each node's data packet and the channel capacity [39]. As such, if the data packet size could be reduced, more nodes could be fit into the network without risking packet collisions. For example, the Neurograins network in [39] can be scaled to 1000 channels at the maximum, but with smaller data packets, a larger number of nodes could be accommodated.

Monolithic implants suffer from the same problem: a finite channel can only communicate out so much data. Eventually, if more information is to be extracted from the brain, bandwidth becomes a constraint and on-node data compression is required.

2.4 Data Compression

There are two main methods for compressing data: lossless and lossy compression.

2.4.1 Lossless Compression

Lossless compression began with Claude Shannon's seminal work [15]. In it he proved that, for any given message, giving shorter codewords to more likely symbols reduces the number of bits required to store the message, without losing any information. This is the basis of lossless compression: more likely symbols get shorter codewords. Lossless compression works best if a subset of values are much more likely than others, e.g. if the data histogram is very narrow and/or skewed, i.e. non-flat. As such, lossless compression generally consists of two steps. The first is pre-processing the data to make certain values more likely. The second is then applying an entropy encoding that assigns shorter codewords to more likely values.

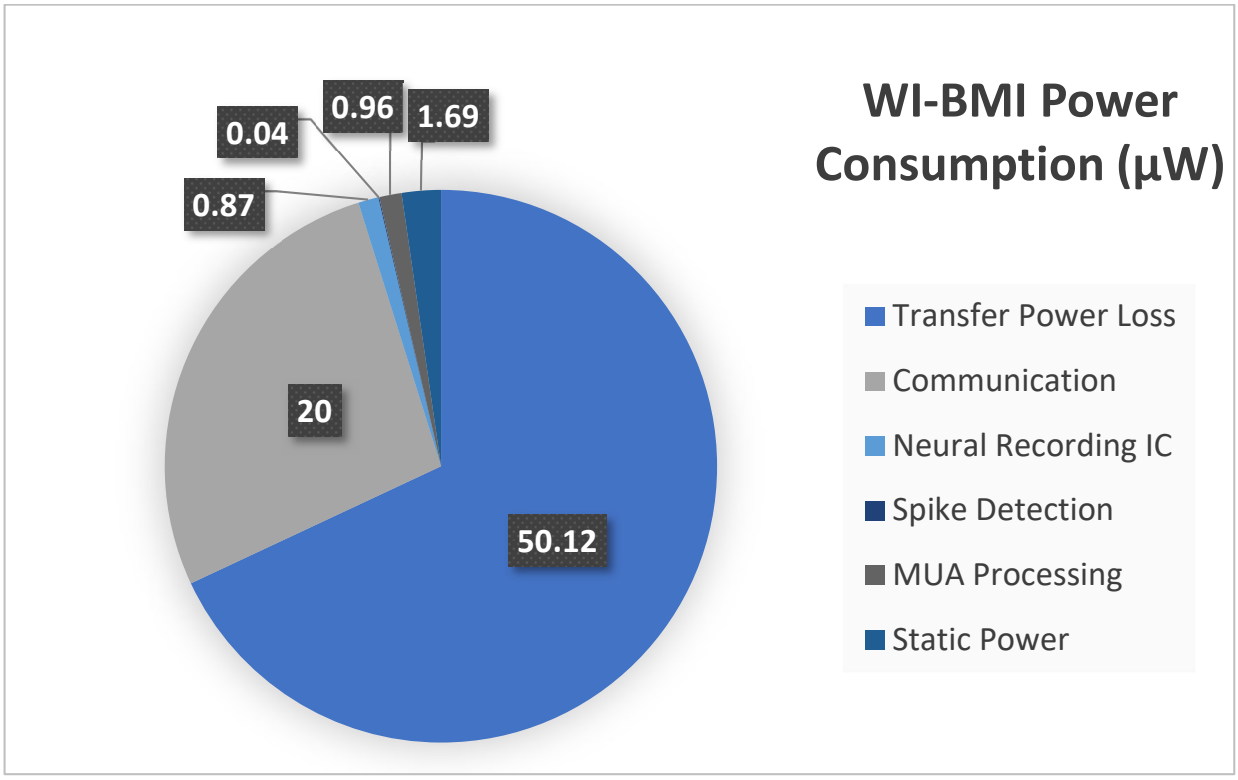


Figure 2.3: Highly optimistic breakdown of WI-BMI power consumption. Each power estimate, in μW, is the lowest estimate for that module that I could find in the literature. It assumes a front-end Neural Recording ASIC with 0.87 μW power consumption [67], an FPGA static power of 162 μW shared across 96 channels (see Chapter 4), a spike detection power of 0.04μW/channel [68], a 0.96μW/channel Multi-Unit Activity (MUA) binning power (to 1 ms temporal resolution) (see Chapter 4), and a 20 nJ/bit communication energy [61], coming to a 20μW communication power assuming a 1 kbps/channel MUA BR (see Section 2.4.2). Finally, the power transfer loss of 50.12μW is calculated as the loss after a 32% power transfer efficiency [50], given as 23.56μW $\times$ (100-32)% / 32%. In reality, the power consumption will likely be much higher, especially due to a higher communication energy and larger transfer losses. However, on-implant recording and processing power may even be reduced, due to an ASIC implementation. Furthermore, the static power calculation assumed 96 channels on-implant, which can of course vary.

2.4.1.1 Pre-processing

The simplest form of pre-processing in time-series data (of which neural data is an example) is delta-sampling. Instead of sending out the current value, one sends out the difference between it and the previous value. This can be generalised to any previous value:

$$y[n] = x[n] - x[n-u] \tag{2.2}$$

for $u \in \mathbb{Z}^+$. If the difference between the subtracted values is likely to be small relative to the previous dynamic range of the data, this narrows the data histogram. However, there are more advanced methods. For example, Linear Neural Network Time (LNNT) sampling uses a learned linearly weighted sum of h past samples to predict the current sample [69]. The prediction error is then communicated off-implant. As long as the weights are known off-implant, the signal can be reconstructed from the prediction error.

LNNT is a generalisation of delta-sampling, where in delta-sampling only 1 past value is used to predict the current value, and the past value is given a weight of 1. LNNT can also be thought of, from a signal processing perspective, as an Finite Impulse Response (FIR) filter that seeks to filter out the signal one is measuring. In this way, the prediction error (i.e. the output of the FIR filter) will be small, i.e. the histogram will be narrow.

In practice, the weights are obtained by training a single-neuron Linear Neural Network on offline training data. The Linear Neural Network finds a linearly weighted sum of h past values that closely estimates the current value. h is chosen prior to training by the researcher. LNNT encoding in a BMI context was first proposed in [69], where LNNT was used to compress 16-bit Extracellular Action Potential (EAP) band recordings sampled at 20 kHz.

2.4.1.2 Entropy encoding

Once pre-processing has narrowed the data histogram, the next step is to use an entropy encoding to losslessly compress the data. Many different entropy encodings have been proposed. These include Huffman encoding, Arithmetic encoding, Lempel-Ziv, etc.

2.4.1.2.1 Huffman encoding

Huffman coding uses a dictionary where each symbol is represented by a unique codeword. Each codeword is optimally close in length to the symbol's Shannon entropy [15] while being independently decodable [70]. As such, it gives optimal compression among methods that encode symbols individually. Huffman compression is especially interesting for WI-BMIs.

Firstly, this is because Huffman encoders can be static. Static encoders are pre-trained offline on representative data and do not adapt to the data they are compressing. As such, if the histogram of the to-be-compressed data can be accurately estimated *a priori*, a static encoder can be used. Relative to an adaptive encoder, this dramatically reduces the required operations and hardware resources because a SH encoder can be implemented using only a few LUTs. An example of Huffman encoder LUTs given in Table 2.2 (a-b), where compression is achieved by giving shorter codewords to more likely symbols. For example, given the codeword lengths and frequencies in Table 2.2 (b), the average encoded length of a symbol will be $0.8 \times 1 + 0.1 \times 2 + 0.07 \times 3 + 0.03 \times 3 = 1.4$ bits, instead of the 2 bits that are normal for the binary representation of 4 values (e.g. codewords of 00, 01, 10, 11).

Secondly, each symbol is represented by a unique, fixed codeword. In particular, Huffman coding is a prefix coding. This means that no codeword is the prefix of another codeword. An advantage of this is that multiplexed channels can use different encoders. As long as the sequence of channel-encoder pairs is correctly

Table 2.2: Demonstration of multiplexed Huffman encoding. (a) Huffman encoder a, trained on equal probabilities. (b) Huffman encoder b, trained on skewed probabilities. (c) Pairings of encoder to symbol sequence to channel. (d) Each symbol can be assigned to any encoder, and the sequence can be fully decoded if the pairings are known. This allows channels to use appropriate encoders from a selection.

(a)

Encoder a Huffman Table		
Symbol	**Frequency**	**Huffman Code**
0	0.25	00
1	0.25	01
2	0.25	10
3	0.25	11

(c)

Symbol - Encoder - Channel pairings		
1st Symbol	Encoder a	Channel 1
2nd Symbol	Encoder b	Channel 2
3rd symbol	Encoder b	Channel 3

(b)

Encoder b Huffman Table		
Symbol	**Frequency**	**Huffman Code**
0	0.8	1
1	0.1	01
2	0.07	001
3	0.03	000

(d)

Encoding and Decoding Signal	
Multiplexed signal (channel 1, channel 2, channel 3)	3; 0; 1
Encoded mult. signal	11101
Decoded mult. signal	From encoder a: 11 = 3; From encoder b: 1 = 0; From encoder b: 01 = 1;

known by the decoder, the sequence can be fully decoded. Considering that multi-channel neural recordings are often multiplexed and can have differently shaped histograms, this can be of great benefit. For example, each channel could use its own encoder. Alternatively, a handful of static Huffman encoders could be placed on-implant. The channels could then be assigned to different encoders based on which offered maximum compression.

As such, Huffman encoders are very simple encoders which offer a lot of flexibility, and are generally well-suited to the WI-BMI environment. An example scenario with two multiplexed Huffman encoders, encoding different channels of neural data, is given in Table 2.2. Channel 1 uses encoder a, channel 2 uses encoder b, etc. (Table 2.2 (c)). The multiplexed signal can be decoded if the symbol-channel-encoder couplings are known by the decoder (Table 2.2 (d)).

2.4.1.2.2 Arithmetic encoding

Another form of compression, considered optimal for multi-symbol compression, is Arithmetic encoding. It works by encoding an entire string of data with a single codeword, based on the probability distribution of each element of the string. An example of an Arithmetic encoder is given in Fig. 2.4.

In terms of hardware efficiency, it can suffer as it typically requires multiple multiplication modules in floating point. However, fixed-point versions that require only a single multiplier have been proposed [71]. They are an interesting avenue for WI-BMI on-implant compression.

2.4.1.2.3 Lempel-Ziv encoding

Lempel-Ziv encoding is another commonly used lossless compression technique. It involves real-time building of a dictionary of incoming symbol-sequences.

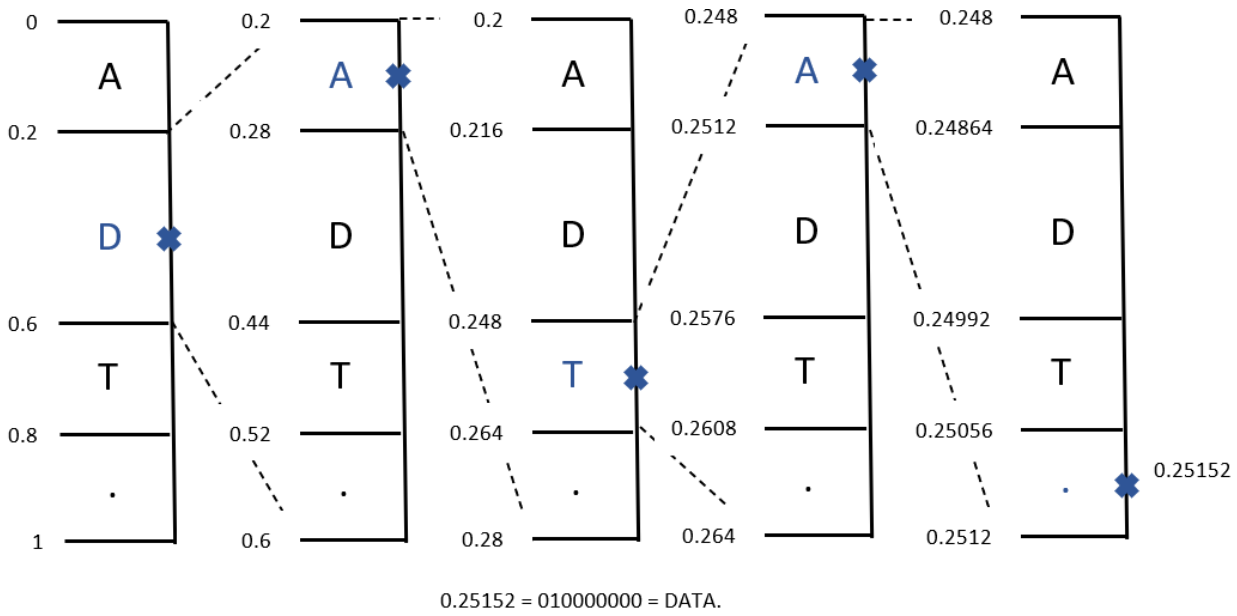


Figure 2.4: Example of Arithmetic encoding.

If a measured symbol-sequence already exists in the dictionary, its codeword is sent out. If it is not already in the dictionary, it is added to the dictionary. A binary example is given by:

$$AABABBBABAABABBBABBABB$$

The data is scanned, and partitioned into novel sequences. 'A' is the first novel sequence, and so is stored in the dictionary. 'AB' is the next novel sequence, followed by 'ABB', followed by 'B', etc.

$$A \mid AB \mid ABB \mid B \mid ABA \mid ABAB \mid BB \mid ABBA \mid BB$$

Each sequence is then stored in the dictionary as a codeword that builds on the earlier dictionary entries. 'A' is the first sequence, so it is encoded in binary as '0' with position 1. Since it is an individual symbol, we add a reference 'x' to the beginning, encoded as '00'. 'AB' consists of reference 'A', which is stored in the dictionary in position 1, and 'B', which is new novel element as so is encoded in binary as '1'. Therefore 'AB' is encoded as (position of reference A) + (binary representation of 'B') = '11', as position 2 (10 in binary). Importantly, the position is encoded with g bits, where $g = ceiling(log_2(position))$. This process is repeated for every element of the sequence, given in Table 2.3. The final encoded sequence is then:

$$000111010010100101111001011001111$$

There are many different implementations of the Lempel-Ziv encoding, but they all require a larger amount of memory. This is because building the dictionary in real time requires a large amount of memory, as well as processing power to continuously access the RAM. As such, they are not appropriate for extremely hardware restrained environments like WI-BMIs. Therefore, they were not further considered in this thesis.

2.4.2 Lossy Compression

The second form of compression is lossy compression, i.e. feature extraction. It consists of compressing data by removing information that is assumed to not be of interest. For example, downsampling data is a form of lossy compression. Similarly, reducing the sampling resolution is a form of lossy compression. In

Table 2.3: Example of Lempel-Ziv encoding dictionary.

Position	1	2	3	4	5	6	7	8	9
Sequence	A	AB	ABB	B	ABA	ABAB	BB	ABBA	BB
Numerical Representation	xA	1B	2B	xB	2A	5B	4B	3A	7
Code (Position of reference + new symbol)	00+0	1+1	10+1	00+1	010+0	101+1	100+1	011+0	0111

neuroscience, it is typical to extract the single-unit FRs by highpass filtering the broadband signal, applying some threshold or wavelet transform for spike detection, applying Principal Component Analysis (PCA) on the spike shape, and finally performing clustering of spike shapes to sort each spike to a putative neuron. This another form of lossily compressing the broadband signal.

More generally, intracortical electrophysiological features come in 2 major classes: data-agnostic and data-adaptive methods. Data-adaptive methods obtain samples and or/statistics on the data, and adapt their transform to the data. A classic example is PCA, which is a common form of pre-processing neural data. It is used in spike-sorting, as well as during dimensionality reduction of spike trains. It works by losslessly projecting the inputted features into a new linear-algebraic basis, where the outputted principal components are orthogonal to each other and ranked by how much of the variance of the inputted features they explain. A means of lossy data compression is then to only communicate out a small number of the principal components that explain most of the variance. However, it is currently infeasible to compute PCA on-implant in WI-BMIs, because of the significant amount of required memory and matrix multiplications during the eigenvalue decomposition. Other well known data-adaptive methods include Independent Component Analysis (ICA) and T- distributed Stochastic Neighbor Embedding (t-SNE), which are both also computationally intractable in WI-BMIs.

Data-agnostic methods involve performing the same transform on the data, regardless of what the data turns out to be. As such, they are typically far more hardware-efficient than data-adaptive methods. In intracortical BMIs, data-agnostic feature extraction can be further split into two major classes: LFP and EAP features. This is because the raw broadband extracellular signal can be split into two major components [28], the LFP and the EAP, both of which are sparse and shown in Fig. 2.5.

2.4.2.1 Local Field Potentials

LFPs consist of the lowpassed broadband at 100-300 Hz, and are believed to result from the sum of extracellular currents and spike activity in the vicinity of the electrode [72,73]. While low bandwidth, easy to measure and chronically available, it has not been shown to have as good decoding performance as higher-frequency features such as those derived from the EAP [16].

2.4.2.2 Extracellular Action Potential

The EAP consists of the approx. 300 Hz highpassed broadband. It is highly sparse, given the infrequency of neural spiking events which are believed to contain most of the interesting information in neural signals. Most of what makes up the EAP signal is spikes of varying amplitude combined with thermal and electronics noise [66,74,75]. To remove the uninteresting noise, many different EAP compressions have become common, e.g. SUA, MUA and ESA. These are shown in Fig. 2.5.

2.4.2.3 Single Unit Activity

SUA consists of measuring the spike firing times of individual neurons in the vicinity of the electrode. It is obtained by identifying neurons firing in the EAP, detectable via a sudden spike in the signal. This is often done via setting a threshold in combination with a nonlinear energy operator, and if the threshold is exceeded then a spike is considered to have occurred. An alternative method is template matching, often done with custom wavelets. Once the spikes in a recording have been detected, the spike shapes are then clustered, where similarly shaped spikes are assumed to originate from the same putative neuron. This is often automated with specialized software, such as Wave_clus [76]. Spike sorting requires daily re-calibration of intracortical BMIs due to micro-motions between the electrodes and neurons that cause drift in spike shape. However, the re-calibration times have gotten to as low as 37 seconds [77].

SUA recordings are generally held to have the highest information content of intracortical neural signal representations. This is because the contributions of individual neurons to the overall signal are explicitly identified [30]. However, the SUA encoding often deteriorates in quality over the course of a few months after implantation of the electrodes. This is due to the foreign body response where the electrodes are encapsulated by fibrous scar tissue [78]. This scar tissue physically distances the electrodes from active neurons, acting as a lowpass filter and making spike sorting difficult. While this can be mitigated by research into material science and non-invasive electrode insertion methods, so far long-term recordings of single-units have been difficult to achieve in free-moving animals [78]. Additionally, SUA is considered to be prohibitively computationally expensive to compute on-implant in WI-BMIs without a computer-to-implant downlink [66, 79]. As such, SUA is highly informative, but unstable and likely too computationally expensive for WI-BMIs. There has been work to extract and compress the spike shapes to send them off-implant for sorting [80, 81].

2.4.2.4 Multi Unit Activity

MUA is similar to SUA, however the spikes are not sorted. The difference is that one treats all spikes on the same electrode as originating from the same putative neuron. Although evidence is somewhat mixed, it is generally believed that MUA gives very similar decoding performance to SUA [16, 82–84]. By reducing the broadband to only the spikes in either SUA or MUA, significant *de facto* compression is achieved, along with power savings [74, 79].

Most modern WI-BMI systems target the MUA signal [74, 77, 84, 85]. This is likely due to its well understood and sparse nature, ease of measurement, and high BDP with a standard BR of at most 1000 bps/channel [64, 74, 86, 87]. It has also been reported that the MUA signal can also be reliably extracted using very few hardware resources and a very small power budget [88]. Unlike the SUA signal, the MUA signal has been reported to be chronically available, as spike detection is significantly more robust than spike sorting.

Since the MUA signal is the premier signal in modern WI-BMI, a significant portion of this thesis will investigate methods for its compression. The typical MUA BR of 1000 bps/channel is because the MUA signal is typically sampled at 1 kHz, at 1-bit sampling [74, 86]. Alternatively said, the spikes in the MUA signal are binned at a 1 ms Binning Period (BP), and the binned values are thresholded at a maximum value of 1. This 1 ms resolution is because spikes typically last some 2 ms [66], and so 1 ms is considered lossless in terms of temporal resolution. However, increasing the MUA BP for data compression is an area of active investigation for intracortical BMIs [74], and will be extensively investigated in this thesis.

2.4.2.5 Entire Spiking Activity

ESA, sometimes also confusingly referred to as MUA, consists of rectifying the EAP and then lowpass filtering it at ∼50 Hz. This gives an envelope of unsorted spiking activity, and has been found to offer high

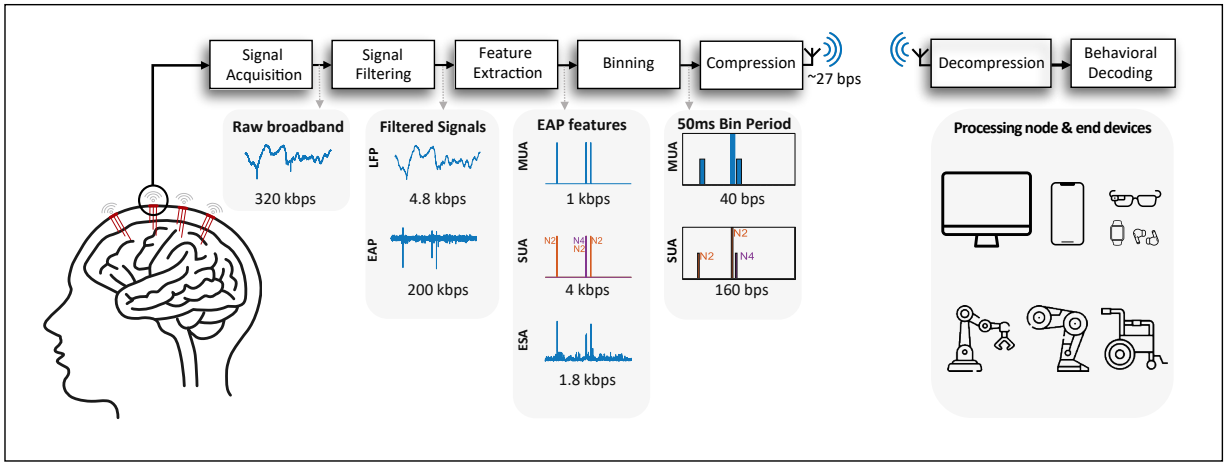


Figure 2.5: Typical BMI data processing and compression flow, with common extracted features / lossy compressions of intracortical broadband data. The numerical values beneath the signals give approximate BRs per channel for that signal. Figure co-created with Zheng Zhang (Z.Z.), co-author of [59], where this figure is taken from.

decoding performance [16, 63, 89, 90]. While SUA and MUA involve the detection of binary events, ESA is more analogue. It is underexplored as a signal, but sampling rates as low as ∼1000 Hz (with a Nyquist rate of 24 Hz) and sampling resolutions of 16 bits/sample offered exceptional decoding performance [16]. By comparing the BDP of SUA, MUA, LFP and ESA signals, across a wide range of decoding algorithms, it was found that the ESA signal had the highest BDP [16]. Furthermore, the ESA signal is chronically recordable, outperforming the LFP, SUA and MUA signals in terms of reduced loss in decoding quality as a function of time [16]. For chronically implanted intracortical BMIs, the ESA is thus an interesting signal. It may be that the sampling rate and resolution could be significantly reduced without degradation in the decoding performance. Non-linear quantisation, and lossless compression of ESA such as in [69], may offer further reductions in BR. This will also be explored in this thesis.

2.5 Neural Decoding

2.5.1 Supervised Machine Learning

Motory BMIs rely on decoding neural activity into behavioral signals. This is done using machine learning. There are three major different kinds of machine learning: supervised learning, unsupervised learning, and reinforcement learning. Which to use depends on one's problem and available data. BMI decoding is typically done using supervised machine learning, since the data is clearly labelled as either neural or behavioral. Specifically, BMI decoding is an example of time-series prediction, which is a form of regression. This is because BMI decoding consists of predicting the behavioral time-series output from neural time-series input.

To do so, firstly, one splits the neural and behavioral data, with matching timestamps, into "training" and "testing". The training data will be used to train the algorithm. The testing data will then be used to test the performance of the final trained application. The second step is to split the training data, again, into what is a bit confusingly referred to as the training and validation data. For the sake of clarity, here we will refer to the data from the 1st split as `train1` and `test`, and the data from the 2nd split as `train2` and `validation`. Fig. 2.6 gives an example of supervised machine learning data flow to decode neural data into behavioural data.

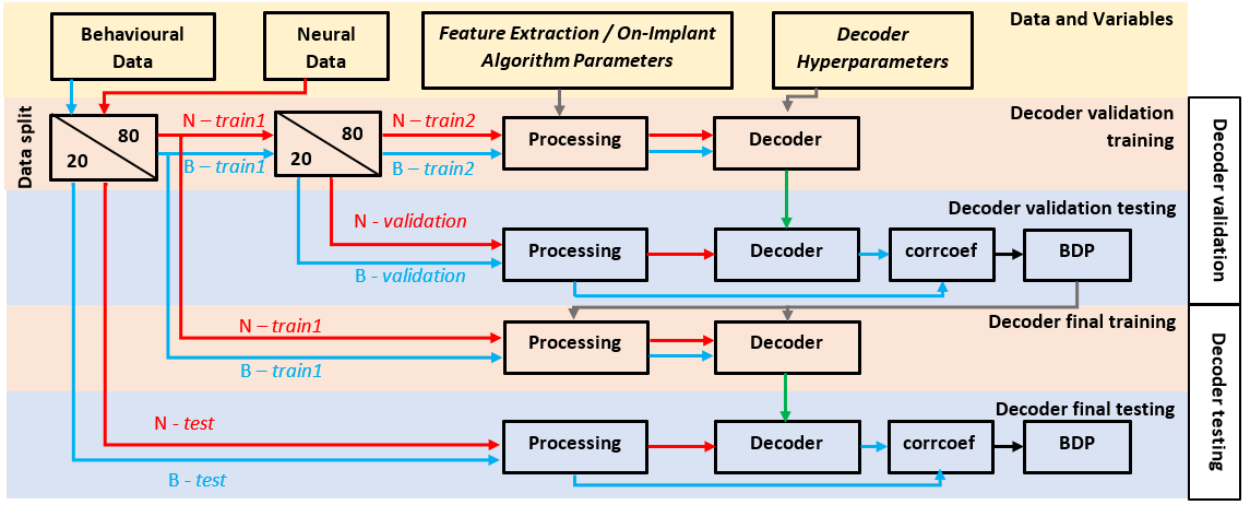


Figure 2.6: Supervised machine learning data flow for BMI decoding problem. corrcoef: Pearson correlation coefficient, here used as the Behavioral Decoding Performance (BDP) metric. Red: Neural data, Blue: Behavioral data; Green: Trained decoder; Grey: Parameters.

The reason for the `train2` and `validation` split is because machine learning / feature extraction algorithms can have a large number of parameters, and the workflow involves determining which parameters will perform the best for the given data. The parameters of a specific machine learning algorithm, e.g. the number of neurons in a Neural Network layer, are referred to as hyper-parameters. Decoder validation, during training, is used to find decoder hyper-parameters and feature extraction / algorithm parameters that give the best decoding result. This is done by trying out many different parameters in parallel when training using the `train2` data, and comparing the decoded results to the ground truth `validation` data.

The best-performing validation parameters are then tested on the separate, so-far-untouched `testing` data. The reason validation is done is to avoid what is called "overfit". This is because there may be noise in the `train1` data that the chosen parameters are over-fitting to, meaning they fit to specific noise in the data and not to general features that will perform well across datasets. That is why we have separate decoder validation and testing: testing gives us an unbiased estimate of our decoder performance.

To mitigate overfit during validation, it is common to use something called k-fold cross-validation. The idea is to split up the `train1` data into k (e.g. 10) non-overlapping segments. Then, for each of the k parallel runs, one of the k segments is used as the validation data, and the other (k-1) segments are used as the `validation` data. For each of the k runs, you try all of the different parameter combinations. You then select the parameters that on average performed best across the k runs. This gives a less biased, more general estimate of what are good parameters, as this is less likely to overfit to noise in any 1 random `train2`/`validation` split. This is not represented in Fig. 2.6, and would correspond to the 2$^{\text{nd}}$ data split. A 80-20% data split for `train1-testing` is common, but arbitrary, and will vary by application.

2.5.2 Neural Decoding Algorithms

Significant work has been done on analysing the performance of different decoders for motory BMIs. In particular, [16, 91] analysed the performance of various classical and Deep Learning algorithms for decoding from various neural features, i.e. SUA, MUA, LFP and ESA. In general, DL decoders such as Long-Short Term Memorys (LSTMs) [92] and Quasi-Recurrent Neural Networks (QRNNs) outperform classical decoders such as Kalman filters, Wiener filters, and Wiener Cascade Filter, but take significantly longer

to train. However, of all classical decoders, Wiener Cascade Filter performed the best. As such, given the significantly shorter training times, this thesis made significant use of Wiener Cascade Filter for neural decoding.

2.5.3 Behavioral Decoding Performance

Throughout this thesis, the X and Y-axis cursor velocities in hand reaching tasks were used as the observed behavioral data. The BDP is defined in this thesis as the across-axes average Pearson correlation coefficient r between the predicted and observed X and Y-axis velocities. The BDP metric is given in Eq. 2.3:

$$\text{BDP} = \frac{\sum_{i=1}^{n}(Vx_i^o - \bar{Vx_i^o})(Vx_i^p - \bar{Vx_i^p})}{2\sqrt{\sum_{i=1}^{n}(Vx_i^o - \bar{Vx_i^o})^2}\sqrt{\sum_{i=1}^{n}(Vx_i^p - \bar{Vx_i^p})^2}} + \frac{\sum_{i=1}^{n}(Vy_i^o - \bar{Vy_i^o})(Vy_i^p - \bar{Vy_i^p})}{2\sqrt{\sum_{i=1}^{n}(Vy_i^o - \bar{Vy_i^o})^2}\sqrt{\sum_{i=1}^{n}(Vy_i^p - \bar{Vy_i^p})^2}} \quad (2.3)$$

where Vx_i^o and Vx_i^p are the observed and decoder-predicted X-axis cursor velocities at sample i, Vy_i^o and Vy_i^p are the equivalents for the Y-axis, and n is the number of samples in the recording.

The Root Mean Squared Error (RMSE) is also a common BDP metric. However, this thesis has not made use of it, since it does not provide any intuitive understanding about the mismatch between the predicted and observed output. Furthermore, the two could be perfectly correlated, but scaled differently, and the RMSE would tell us that the signals are mismatched when it is merely a question of gain. As such, this thesis uses the Pearson correlation coefficient as the unique BDP metric.

2.5.4 Behavioral Temporal Resolution

Another metric to consider during decoding is BTR. The decoder may achieve high BDP, but if the decoder only outputs decoded hand velocities at a BTR of 10 s intervals, that is unacceptable for most decoding applications. BTRs in the literature vary, but a BTR 100 ms or less is typical [64, 74, 77, 86, 87]. In [93], it was shown that the mean human reaction time in 120 healthy 18-20 medical students was larger than 220 ms for both auditory and visual stimuli. Therefore, it may be that a BTR of 100 ms may be well tolerated for motor decoding applications in terms of delay in user experience.

However, there is some concern that a BTR of 30 ms or smaller should be prioritised, since it can improve the fluidity of the user feedback loop. A 100 ms pause each time a cursor moves is not a fluid experience for the user, and so there is good reason to believe that a smaller BTR should be prioritised. However, if smoothness of the user feedback loop is what needs to be prioritised, it may be that adding a smoothing function to a controlled cursor, for example, may solve the problem, enabling larger BTRs. The ideal BTR for hand kinematics has not been identified in the literature, and so this thesis will treat the ideal BTR agnostically, while remaining below 100 ms.

2.6 ASIC vs. FPGA

ASIC designs are optimal in terms of minimising power consumption and chip area. However, the design process of FPGAs is significantly less expensive and easier compared to that of ASICs. FPGAs also benefit from increased flexibility for programming, which accommodates rapid testing of algorithmic changes better. FPGA results are also a good approximation of the ultimate ASIC design [57, 58, 94]. As a result, it is

typical for researchers to use FPGAs to validate the ASIC's performance before full development of the ASIC [58, 95].

That is the approach adopted in this thesis: we consider systems implemented in reconfigurable hardware FPGA. In particular, we make extensive use of the Lattice ice40LP FPGA board as an example target for our WI-BMI. This is because it is an ultra-low power, high-performance FPGA with 40 nm technology and small BGA package that is ideal for the thinnest devices like implantable BMIs. The Lattice ice40LP1K in particular is an ultra-low-power FPGA board with 1280 logic cells and sixteen 4kbit bRAM memory blocks.

Therefore, to determine the power consumption of the different algorithms considered throughout this thesis, the algorithms were simulated on the Lattice ice40LP. This simulation was done to assess the overhead on power and resources brought by compression to guide our configuration selection. All programs are written in Verilog, simulated on Mentor Modelsim Lattice Edition, and synthesized with iCECube 2020.12.

The resource occupation was obtained from the Placing Summary of ICEcube2, and indicates the number of LUTs and FFs used. To get the power consumption, it is not practical, given the massive parameter space, to download all the different configurations to the FPGA board and measure their power consumption individually. The iCEcube2 power estimator was used to estimate the processing power to reduce the assessment load. Although the estimation is not the real power consumption, it is accurate enough for estimating the effect of different configurations and guiding the selection process.

Ultimately, any design for WI-BMIs should be implemented in ASIC hardware, for reductions in power and chip area [57, 58, 94]. As such, the FPGA results throughout this thesis should be interpreted as guiding the selection process for an eventual ASIC design. The FPGA power and resource occupation simulation results are sufficient to allow us to make a decision on a well-performing FPGA system. However, when interpreting the FPGA results to guide the ASIC architecture, some nuances should be observed.

The four metrics that we judge our compression work on are power consumption, hardware resources, behavioral decoding performance and temporal resolution. The key discrepancy between ASIC and FPGA will be the trade-off between processing and communication power, and so the results of this thesis should be read with that in mind.

As shown in Fig. 2.3, the single largest aspect of power consumption is power transfer loss. It is not obvious how that should be accounted for: should it be considered as part of the heating budget, or not? The heating via transfer loss is significantly more diffuse than the heating caused by the on-implant power-consumption, and so how to account for it in the power budget is not clear. In either case, we can be very confident that the communication power reduction from FPGA to ASIC is less impressive than the reduction in processing power. In Table 2.1, we see that ASIC solutions are much more power efficient for communication than FPGA solutions. However, these measurements were made outside of the lossy neural medium, whereas it is likely that a minimum communication power will be needed to traverse the lossy medium. Given that, the difference between FPGA and ASIC communication energy may be quite small, as the dominating factor in communication power is hardware-independent. As such, it is likely that the role of communication power in ASIC will be even more important than shown for FPGAs in Fig. 2.3.

As such, when implementing an algorithm in ASIC, it would likely be best to prioritize lowering the BR, and therefore communication power, even at the expense of lowering the processing power. Processing power in ASIC is likely to be negligible, and therefore more-hardware intensive solutions, if they give compression gains, are likely to be more attractive in ASIC than they would be in FPGA.

As discussed in Section 2.3, this preference for lowering communication power over processing power should also true to a lesser extent when interpreting the FPGA results in this thesis. The 20 nJ/bit FPGA communication energy used throughout this thesis is a very low estimate of what is achievable *in vivo*. Therefore, when considering power efficiency, even in FPGA one should prioritize more significant compression over lowering processing power. We use a low estimate of communication power to be conservative about

the utility of hardware-efficient compression in WI-BMIs.

2.7 Datasets

To get a broad sample of neural recording conditions, five publicly available intracortical recording datasets were used throughout this thesis. These are summarised in Table 2.4, and further detail are given in Appendix A.

Table 2.4: Dataset summaries.

Dataset	Neural data type	Species, electrode type and brain region	Details	Behaviour
Flint [64]	SUA	Rhesus macaque monkey Utah array M1	One subject 12 recordings across 5 days 96 channels Recording lengths (quartiles, s): 597, 604, 630	Free-reaching hand task Continuous data stored
Brochier [96]	SUA	Rhesus macaque monkeys Utah array M1 and PMv/PMd	Subjects N and L Single session recordings 96 channels Recording lengths (s): N: 1003 L: 709	Hand reaching task Target stored
Sabes Processed [65]	SUA	Rhesus macaque monkeys Utah array M1 and S1	Subjects Indy and Loco 37 recordings for Indy across 10 months 10 recordings for Loco across a month 96-192 channels Recording lengths (quartiles, s): Indy: 472, 524, 816 Loco: 1771, 1928, 2384	Free-reaching hand task Continuous data stored
Sabes Raw Broadband [65]	Broadband	Rhesus macaque monkeys Utah array M1	Subject Indy 30 recordings across 10 months 96 channels Recording lengths (mean, s): Indy: 520	Free-reaching hand task Continuous data stored
Jackson and Hall [97]	Broadband	Rhesus macaque monkeys Microwire array M1	Subjects Dusty, River and Silver 64 recordings 10-24 channels Recording lengths (quartiles, s): 364, 396, 473	Not Accessed

M1: Primary Motor Cortex; PMv: Ventral Premotor Area; PMd: Dorsal Premotor Area; S1: Primary Somatosensory Cortex; Utah array: A type of 100-channel microelectrode array (Blackrock Microsystems, Salt Lake City, UT).

Part I

Compressing Multi-Unit Activity

Chapter 3

Static Huffman Compression of Intracortical Neural Signals

This chapter addresses the need for hardware-efficient compression of neural signals in WI-BMIs. This chapter introduces the use of Static Huffman (SH) encoders to solve this problem. It investigates the compression performance of SH encoders for compressing LFP, EAP and ESA signals at various sampling resolutions.

This chapter has been adapted from the following published article:

Oscar W. Savolainen and Timothy G. Constandinou. "Lossless compression of intracortical extracellular neural recordings using non-adaptive huffman encoding." 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC). IEEE, 2020. [98]

3.1 Background

SH encoders are an interesting lossless compression technique for WI-BMIs. They have not been explored previously in WI-BMIs, but have significant benefits (refer to Background, Section 2.4.1.2.1 for details). As is typical in lossless compression, they can also be combined with various pre-processing techniques to improve the compression.

This chapter describes a preliminary study to show that intracortical signals could be effectively compressed with SH encoders [98]. Four different SH paradigms were considered: 1st and 2nd order SH encoding [70], Delta SH encoding, and LNNT SH encoding [69]. Maximum codeword-length limited versions are also considered to protect against overfit to training data. The considered intracortical signals are the EAP signal, the ESA signal, and the LFP signal. Sample resolutions of 5 to 13 bits are considered.

The EAP, ESA and LFP signals were considered in this chapter because they have either been used directly for Behavioral decoding [90] [99], or for the extraction of other signals often used in Behavioral decoding, e.g. SUA [66], MUA [99]. Broadband signals were not considered. This is as they can be more effectively broken into their LFP and EAP components [28]. MUA signals require more pre-processing than these signals for effective compression, and so MUA was not considered in this preliminary study, and will be dealt with in the following chapters.

This chapter also builds on the work done in [69] and [100] to expand the application of LNNT to

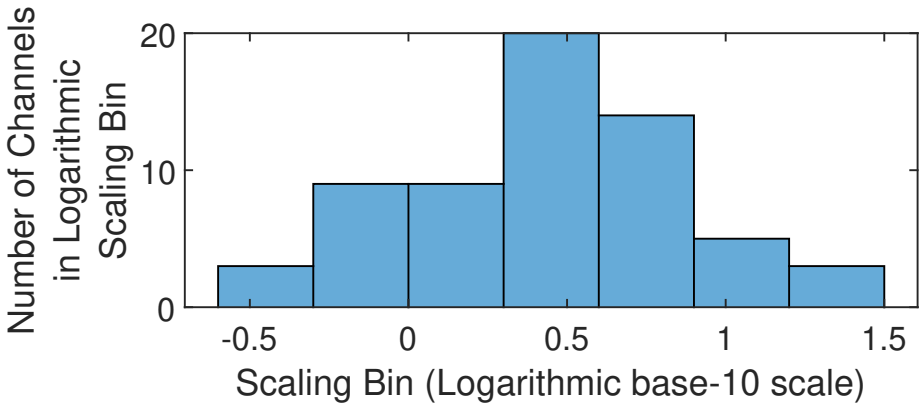


Figure 3.1: Histogram of values used to scale segments prior to concatenation. The values are centered at 100 μV.

different intracortical neural signals at different sample resolutions. It also pioneers the use of limiting the maximum-codeword length in SH encoders via clipping of the probability distribution.

3.2 Methods

All work was done in MATLAB R2019A, using neural recording data from Andrew Jackson and Thomas Hall [97], detailed in Table 2.4.

3.2.1 Signal Pre-processing

All the neural signals were produced from the same intracortical extracellular neural recordings. The recordings originated from [97], and were taken from NHP M1 (Animals: Dusty, Silver and 'Ukiah'). The recordings consisted of Broadband data sampled at 24.4 kHz at 16-bit precision using microwire arrays. In this work, 50 s segments from 63 channels across 25 recordings were concatenated together to produce a single recording of length 3150 s. This is so as to represent a wide range of microwire recording conditions.

3.2.1.1 Scaling signals

LFPs and EAP band signals normally range from 10 μV to 1 mV in amplitude [67]. As such, the peak-to-peak amplitudes of neural signals vary by roughly 2 on a base-10 logarithmic scale. Prior to concatenation, each segment's amplitude was individually scaled to a random value from a normalised logarithmic distribution ($\mu = 1$, $\sigma = 1$). This ensured that the varying amplitudes observed in neural recordings were realistically represented in the used data. The largest scale value was 104 times the smallest. A histogram of the scaling values is given in Fig. 3.1. No other processing was performed on the Broadband data.

3.2.1.2 EAP

The EAP signal was obtained by highpass filtering the Broadband signal at 300 Hz. It was then lowpass filtered at 3.5 kHz to avoid aliasing, and then downsampled to 8.1 kHz. EAP signals sampled at 7 kHz have been found to enable SUA extraction [66]. Thus, the EAP signal was downsampled by an integer factor of 3 from 24.4 kHz to avoid any distortion via interpolation.

3.2.1.3 ESA

The ESA signal was obtained by taking the absolute value of the EAP signal, and then lowpass filtering the result at 50 Hz. The ESA was then downsampled to 200 Hz.

3.2.1.4 LFP

The LFP was taken by lowpass filtering the Broadband at 300 Hz, and downsampling to 1.2 kHz.

3.2.1.5 Miscellaneous

All filters used were digital 3$^{\text{rd}}$ order Butterworth FIR filters, designed using the `designfilt` function. The signals are shown in Fig. 3.2. The first 30% of the signal was used as training data and the remaining 70% as testing data. The training data was chosen as to be somewhat mismatched with the testing data. This is so as to observe the encoders' performances in unexpected environments.

The sample resolution b was varied between 5 and 13 bits inclusive for all of the signals. Therefore, post-processing, each signal was scaled to between 1 and 2^b at its given resolution b so that each sample value was an integer.

3.2.2 Training SH encoders

The dynamic range of the encoders were set to equal that of the concatenated signal. This is practically unrealistic due to saturation concerns. However, this was done to avoid artificially boosting the Compression Ratio (CR) by compressing empty symbol-space, as is unfortunately common with CR measurements. As such, this chapter's key findings are the performances of the encoders relative to eachother, and relative to the ideal CRs gained from the entropy of the testing data.

3.2.2.1 1$^{\text{st}}$ order encoder

In this work, 1$^{\text{st}}$ order SH encoding is based on the 1$^{\text{st}}$ order Shannon entropy. This is representative of the sample values' frequency of appearance in the sequence [15]. The frequencies were calculated for each symbol α within the training data x, where $\alpha \in \mathbb{Z}$ and $1 \leq \alpha \leq 2^b$.

3.2.2.2 Delta encoder

Delta 1$^{\text{st}}$ order SH encoding works along the same principle as 1$^{\text{st}}$ order encoding. The only difference is that the data was delta-sampled prior to encoding.

3.2.2.3 2$^{\text{nd}}$ order encoder

2$^{\text{nd}}$ order SH encoding is based on the conditional probabilities between adjacent samples. This exploits redundant information in the time domain. As such, it gives a codeword to the current sample that is dependent on the value of both the current and previous sample, i.e. a 1$^{\text{st}}$ order Markov source.

The probabilities of any symbol β following any symbol α within each sequence x were calculated, where $\{\alpha, \beta\} \in \mathbb{Z}$ and $1 \leq \alpha,\ \beta \leq 2^b$. A set of 2^b Huffman dictionaries were then created, one for each symbol in α, with 2^b codewords each, one for each symbol of β. The same codewords are used within each α dictionary, but represent different β values.

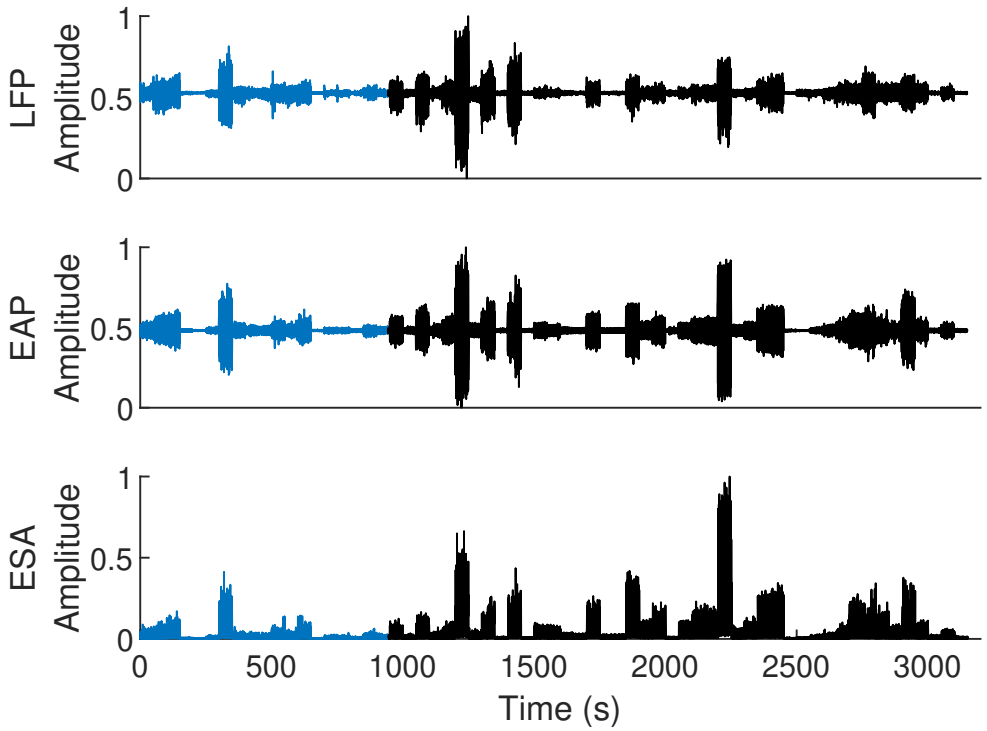


Figure 3.2: Produced signals, with amplitude normalised between 0 and 1. Blue represents the training data and black represents the testing data.

3.2.2.4 LNNT encoding

The LNNT prediction error er_k is a linear sum of the current sample value and past h values in the training sequence x, using weights W:

$$er_k = \sum_{i=1}^{h} W_i x_{k-i} \tag{3.1}$$

The signal can then be reconstructed off-implant:

$$y_k = \sum_{i=1}^{h} y_{k-i} W_i - er_k \tag{3.2}$$

The LNNT predictor weights were trained using $h \in]1:10[$ past samples, and MATLAB's `trainNetwork` function. The LNNT predictor was paired with a 1^{st} order Huffman encoder to compress the prediction error. The LNNT predictor was trained using only $\sim 1\,\text{s}$ of the training data, and with 2000 epochs [69]. The rest of the training data was fed through the predictor. Then the resulting er signal was used to train the Huffman encoder.

3.2.3 Maximum codeword length limitation

A significant issue with pre-trained encoders is overfit to training data. In the worst case, with a large dynamic range, if a value does not occur in the training data it can be given a very large codeword. If it occurs sufficiently often in the testing data, the entire compression will be ineffective. A way to defend against overfit is to limit the maximum codeword size. This gives non-ideal codeword sizes based on the training data, but mitigates the effects of mismatch between training and testing data. As such another set of encoders was trained, where the maximum codeword length was limited to $c = 5$ bits of the unencoded

word length. For example, a sample resolution of 5 bits would have a maximum encoded codeword length of 10 bits. The choice of c was predicated on work done in [101], which suggests that the average compressed word length increases at most by $R \approx 1/(2^c)$ bits.

However, this seems empirically to not be true. R seems rather to represent the amount of probability that is transferred across the distribution in the worst case. As such, the choice of c is perhaps arbitrary and warrants further research. The codeword length was limited by assigning each probability a minimum value of $1/(2^{b+c})$, and subtracting the sum of added probabilities from the most occurring value. In the worst theoretical case this was $R \approx 0.03$, deemed acceptable. After the distributions were edited, the training of the encoders was as in Section 3.2.2.

3.3 Results

The CRs of the encoders were then tested with the testing data, where in each case the testing data was pre-processed in the expected way. Additionally, the encoders' successful decompression of the compressed testing data was verified. The achieved CRs for each signal are given in Tab. 3.1 (p. 4). They are dependent on the sample resolution, compression method, and in the case of LNNT for the number of past samples used. The CRs were obtained by dividing the product of the original sample resolution and the length L of the sequence by the length of the encoded signal.

Also included in Tab. 3.1 is the correlation between h and the resulting LNNT CR. It is only included if the value was found to be statistically significant ($p < 0.05$). Finally, the ideal CRs derived from the testing data entropies are given for each case (Ideal CR = b/Entropy). They are not given for LNNT encoding, due to unknown ideal weights for each case. These give the ideal performance of the encoders where the testing data is compressed down to its entropy. This ideal performance depends on the encoding type, e.g. Ideal 2^{nd} order SH encoding uses the 2^{nd} order Shannon entropy, Ideal Delta 1^{st} order SH encoding uses the 1^{st} order Shannon entropy of delta-sampled data, etc.

The results indicate that, firstly, that maximum codeword-limited encoders universally closely match their regular counterparts at lower sample resolutions, and outperform them at higher sample resolutions. However, the sample resolution of divergence varies by encoding method and signal. The stronger performance of maximum codeword-limited encoders is likely due to their protection against overfit, which is a greater problem at higher sample resolutions due to a larger dynamic range. Maximum codeword-limited encoders often have CRs close to the ideal based on the testing signal's entropy. This highlights the value of overfit-protection in pre-trained encoders.

Secondly, we find that different encoders work best for different signals and sample resolutions. For example, LFP signals have a larger amount of low-frequency content, meaning the previous value will be more predictive of the current value. Accordingly, the clipped Delta 1^{st} order SH, Delta 1^{st} order SH and clipped 2^{nd} order SH encoders performed on average the best for the LFP signal. However, at higher sample resolutions and with signals with higher frequency content (e.g. ESA, EAP), encoders that rely on the previous sample value to predict the current value, e.g. clipped and non-clipped 2^{nd} order SH encoding, perform less well than their peers.

3.4 Discussion

3.4.1 Hardware Efficiency of 2^{nd} order SH encoding

Having 2^b dictionaries, each with 2^b codewords, is untenable in resource constrained environments such as WI-BMIs. However, there is significant redundancy within these codewords between dictionaries, and

Table 3.1: Achieved and Ideal Compression Ratios (CR) for Lossless Static Huffman Encoders Applied to Different Intracortical Neural Signals, at Various Sample Resolutions.

Signal	**Bits**	**Compression Ratio (CR)**														
		SH1	**CSH1**	Ideal SH1	**DSH1**	**CDSH1**	Ideal DSH1	**SH2**	**CSH2**	Ideal SH2	**LNNT_1**	**LNNT_10**	Corr (h,CR)	**CLNNT_1**	**CLNNT_10**	Corr (h,CR)
LFP	5	3	3	3.2	**4**	3.9	6	**4**	**4**	6.8	1.7	1.7	x	1.5	1.6	x
”	6	2.5	2.5	2.6	**4**	3.8	4.6	3.9	**4**	5.1	2	2.2	x	1.9	2.2	x
”	7	2.1	2.1	2.2	3.4	3.3	3.5	3.1	**3.5**	3.8	2.2	2.4	x	2.1	2.4	x
”	8	1.8	1.9	1.9	2.6	2.6	2.8	2.2	**2.9**	3	2	2.1	x	2	2.1	x
”	9	1.6	1.7	1.7	2.2	2.2	2.4	1.3	**2.4**	2.6	2	2.1	x	2	2.1	x
”	10	1.5	1.6	1.6	1.9	2	2.1	0.62	**2.1**	2.2	1.9	1.9	x	2	2	x
”	11	1.3	1.5	1.5	1.5	**1.9**	1.9	0.21	1.8	2	1.5	1.6	0.74	1.8	1.8	x
”	12	0.99	1.4	1.5	1.1	**1.7**	1.8	0.06	1.6	1.9	1.1	1.2	0.65	**1.7**	**1.7**	x
”	13	0.65	1.4	1.4	0.68	**1.6**	1.7	0.02	1.4	1.8	0.68	0.81	0.89	**1.6**	**1.6**	x
EAP	5	3.2	3.1	3.7	2.9	2.8	3.2	**3.3**	**3.3**	4.3	1.6	1.6	x	1.5	1.5	x
”	6	2.6	2.6	2.7	2.4	2.3	2.5	2.7	**2.8**	3	1.7	1.9	x	1.7	1.8	x
”	7	2.1	2.2	2.3	1.9	2	2.1	2.2	**2.4**	2.5	1.6	1.8	x	1.7	1.8	x
”	8	1.9	1.9	2	1.7	1.8	1.9	1.7	**2**	2.1	1.6	1.7	0.77	1.7	1.7	x
”	9	1.7	1.7	1.8	1.6	1.6	1.7	1.1	**1.8**	1.9	1.5	1.6	0.89	1.6	1.7	0.77
”	10	1.5	1.6	1.6	1.4	1.5	1.6	0.62	**1.7**	1.8	1.4	1.5	0.94	1.5	1.6	0.82
”	11	1.3	**1.5**	1.6	1.2	**1.5**	1.5	0.26	**1.5**	1.7	1.2	1.3	0.94	**1.5**	**1.5**	0.87
”	12	1.1	**1.4**	1.5	0.94	**1.4**	1.5	0.08	**1.4**	1.6	0.93	1	0.9	**1.4**	**1.4**	0.89
”	13	0.78	**1.4**	1.4	0.61	**1.4**	1.4	0.02	1.3	1.5	0.6	0.63	x	**1.4**	**1.4**	0.93
ESA	5	2.2	2.2	2.3	2.3	2.3	2.4	2.2	**2.6**	3.1	1.5	1.3	-0.91	1.5	1.4	-0.86
”	6	1.7	1.8	2	1.9	1.9	2	1.7	**2.3**	2.6	1.2	1.1	-0.76	1.3	1.2	-0.76
”	7	1.4	1.7	1.8	1.6	1.7	1.8	1.1	**2**	2.2	1.2	1.4	x	1.4	1.5	x
”	8	1.2	1.6	1.7	1.4	1.6	1.7	0.56	**1.8**	2	1.1	1.3	x	1.4	1.6	x
”	9	0.86	1.5	1.6	1.2	1.5	1.6	0.22	**1.6**	1.8	0.79	1.2	0.85	1.4	1.5	0.81
”	10	0.56	1.4	1.5	0.91	1.4	1.5	0.07	1.4	1.7	0.54	0.8	0.9	1.4	**1.5**	0.86
”	11	0.31	1.3	1.4	0.59	**1.4**	1.4	0.02	1.2	1.7	0.31	0.54	0.91	1.3	**1.4**	0.82
”	12	0.15	1.3	1.4	0.32	1.3	1.4	0.01	1.1	1.7	0.16	0.28	0.97	1.3	**1.4**	0.79
”	13	0.07	**1.3**	1.3	0.13	**1.3**	1.3	0.00	0.95	1.8	0.08	0.13	0.98	**1.3**	**1.3**	0.8

SH1: 1st order Static Huffman encoding; DSH1: Delta-sampled 1st order Static Huffman encoding;

SH2: 2nd order Static Huffman encoding; The C-prefix, e.g. CHS1, signifies the probability distribution the data was trained on was clipped.

‘Ideal’ signifies a CR estimate based on the entropy of probability distribution used to train the SH encoder.

Corr(h,CR)=Correlation between increasing h and LNNT encoding CR. x values indicate correlation measurements that were statistically non-significant ($p > 0.05$). Together, these tell us how much increasing the LNNT h size improves compression performance.

CRs below 1 indicate unsuccessful compression, where the encoded sequence is longer than the original. The best achieved CRs are given in bold.

therefore the size of the total 2^{nd} order encoder can be reduced dramatically. However, it must be kept in mind that the size of 2^{nd} order SH encoders is a serious negative. Additionally, the compression performance of 2^{nd} order encoders is not significantly better than other encoders for any given signal and sample resolution. Therefore, we would argue that the use of 2^{nd} order SH encoders is not warranted in WI-BMIs, at least in the case of ESA, EAP or LFP signals.

3.4.2 Choice of encoding

It is anticipated this chapter can inform the choice of compression method for ESA, EAP and LFP signals in intracortical BMIs. For LFP signals, the results suggest that the clipped Delta 1^{st} order SH encoding encoding is best. For EAP signals, the clipped 1^{st} order encoding performed the best, compressing to near the 1^{st} order entropy. For ESA signals, the clipped 1^{st} order, clipped Delta 1^{st} order and clipped LNNT encoders performed virtually the same at higher sample resolutions. At lower sample resolutions, the clipped LNNT encoder performed less well, likely due to maladaptive LNNT weights.

3.4.3 LNNT encoding

It is worth noting, for the EAP and ESA signals, that at sample resolutions of 9 bits and above, increasing the number of past samples in clipped LNNT encoding statistically improved the performance. The upper limit of how many past samples it is beneficial to add is not known.

It is also worth noting that the LNNT encoding, as a whole, did not outperform the Delta encoding. This is counter-intuitive, since the LNNT is simply a generalisation of Delta encoding. A likely explanation is that using only 1 s of data to train the LNNT weights was not sufficient. The motivation for using so little was to have sufficient data to train the encoder, but with hindsight this was unnecessary. It would likely have been more effective to use all of the train data to train the LNNT weights, and then pass the data through the LNNT pre-processing again, which could then be used to train the SH encoder.

Therefore, the results from this chapter are insufficient to state that LNNT encoding is inferior to delta-sampling, particularly as LNNT is in theory more powerful. As such, it still warrants consideration, but care should be taken to use a sufficient amount of LNNT training data.

3.5 Conclusion

Various SH encoder configurations were investigated to compress EAP, ESA and LFP signals, and the results analysed. The result show that overfit-protection dramatically improves compression, especially at larger dynamic ranges. This is because it protects against mismatched training and testing data. In particular, it protects against sparse values in the training data being more present than expected in the testing data. Across signals, 2^{nd} order encoding generally performed best at lower sample resolutions, and 1^{st} order, Delta and LNNT encoding performed best at higher sample resolutions. However, 2^{nd} order SH encoding is largely untenable due to hardware constraints. It did not significantly outperform the other encodings to the point that it justified its extra hardware costs. Having determined that SH encoding is effective for compressing various intracortical signals, the next step involves seeing if the same is true for MUA signals, given some extra pre-processing.

The proposed SH methods should also generalise to other remote sensing applications, of which WI-BMIs are a subset. SH encoders are appropriate for any remote sensing application where the distribution of the sensed data can be estimated *a priori*.

Chapter 4

Static Huffman Compression of MUA

This chapter builds on the work in Chapter 3 by applying SH encoders for the compression of MUA. We performed a holistic analysis, looking at the effects of MUA data compression on BTR, BDP, hardware requirements and total on-implant power. We achieved extreme hardware efficiency and compression rates for MUA data that were at least an order of magnitude greater than the standard rate of 1000 bps/channel, with little to no reduction in BDP. This work was done in collaboration with Zheng Zhang (Z.Z.), a peer and PhD student at the Next Generation Neural Interfaces (NGNI) lab, Imperial College London, who contributed the hardware design and optimisation work.

This chapter has been adapted from the following published article:

Savolainen, Oscar W., et al. "Hardware-Efficient Compression of Neural Multi-Unit Activity." IEEE Access, 2022.

4.1 Background

In the previous chapter, we developed SH methods for the compression of the EAP, ESA and LFP intracortical signals. It was a preliminary work to investigate SH encoders for intracortical data. However, the MUA signal was not immediately considered, as it requires further pre-processing, affecting its BTR and BDP. Given the success of SH encoders for other intracortical signals, this chapter tackles the MUA signal. As such, this chapter performs a holistic analysis of compressing MUA signals with SH encoders using a 'windowed' method, looking at the effects of compression on BDP, BTR, hardware resources and total on-implant power.

4.1.1 Standard Windowed MUA representation

Multi-channel MUA is typically represented as multiplexed data. The length of the data block is $n \times m$, where n is the number of channels and m is the number of bits used to represent the number of MUA events per BP on each channel. An example using a standard binary representation with m and $n = 3$:

001 111 000

would indicate that 1 neuron fired (001) on channel 1, 7 neuron firings occurred (111) on channel 2, and 0 (000) on channel 3. An advantage is that the channel ID is implicitly encoded in bit position, and so does

not need to be explicitly encoded. E.g.,

$$c = ceil(t/m) \tag{4.1}$$

where $c \in [\mathbb{Z}, 1 \leq c \leq n]$ is the channel ID, $t \in [\mathbb{Z}, 1 \leq t \leq n \times m]$ is the bit position and *ceil* is the ceiling function. m is the number of bits required to represent all possible MUA FRs losslessly, and is generally set as $ceil(log_2(\texttt{max}(X) + 1))$, where X is the multi-channel MUA data.

While this does not have a standard name in the literature, in this thesis we refer to it as the "windowed encoding". This is because it sends out the number of MUA events in a non-overlapping window of length BP for each channel, regardless of whether an event occurs on a channel or not. It is ubiquitous throughout MUA BMI work [74, 86] since it is easy to implement and interpret.

As such, the windowed encoding, in it simplest form, has a BR of:

$$\text{BR} = \frac{m}{\text{BP}} \quad \text{[bps/channel]} \tag{4.2}$$

As discussed in the Background Section 2.4.2.4, MUA signals typically use a BP, i.e. temporal resolution, of 1 ms. At a 1 ms BP with 1-bit sampling ($m = 1$), this corresponds to a standard BR of 1 kbps/channel.

4.1.2 Prior Work in the Compression of MUA

It was proposed in [74, 102] that increasing the MUA BP from the standard 1 ms may be an efficient way to lossily compress MUA data. This will be investigated in this chapter. However, this compression is lossy because increasing the BP reduces the temporal resolution of the MUA firing times. This could cause two problems.

The first is increased delay in BMI-user experience: the BTR. When we increase the BP, we increase the maximum possible lag between a neuron firing and the data being communicated off-implant. In this system the BP is equal to the BTR, since the BP is the temporal bottleneck of the system. There are no other significant delays in the MUA WI-BMI data flow other than the spike counting, as the communication and other processing of the data occur on the ns scale. As such, increasing the BP increases the BTR, which should probably be kept at 30 ms or lower, and almost certainly at less than 100 ms as discussed in Section 2.5.4. In this chapter, we will investigate BPs, and therefore BTRs, between 1 and 100 ms.

The second potential issue with increasing BP is reduction in BDP. The reduced temporal resolution of neural firing times may reduce our decoding ability. [91] found that, for SUA signals, there was no difference in hand kinematic decoding ability between BPs of 10-100 ms for LSTM, Feedforward NN, and Wiener filter neural decoders. However, they also found that, when using Kalman filter decoders, increasing the SUA BP up to 50 ms improved the decoding, although not to the level of the NN decoders. Additionally, a 100 ms BP for motor decoding is a common choice by researchers [64, 87, 91].

As such, the effect of MUA BP on BDP is not clear, and it has been hypothesised that it likely varies by decoding algorithm and decoded task [74]. The effect of BP and limiting the dynamic range of MUA data on compression and BDP are one important aspect that will be further investigated in this chapter.

4.1.3 This Work

This chapter proposes and compares multiple hardware efficient windowed MUA compression schemes. To the best of the author's knowledge, it represents the first study on compressing MUA signals. It also evaluates how using one or multiple SH encoders, saturating the dynamic range, using on-implant histograms to add adaptivity into SH encoders, and setting different BPs can compress the data and affect the behavioral decoding quality. The goal of this chapter is to seek the best MUA compression algorithm with minimal

resources to reduce BR so as to reduce the on-implant power (processing and communication power) without degrading BDP, while also keeping the BTR within an acceptable range. The original contributions of this chapter are summarised below:

- Empirically showing the degree to which increasing MUA BP lossily decreases the communication bandwidth.
- Limiting the dynamic range of MUA data to reduce the communication bandwidth.
- The use of SH encoders for the compression of MUA data.
- The use of a sample histogram with mapping to add adaptivity to SH encoders.
- The use of multiple SH encoders, with assignment via a sample histogram, to add adaptivity to SH encoders.
- A novel machine learning algorithm for the offline selection of the best combination of SH encoders.
- A holistic analysis of the effects of MUA data compression on total implant power, BDP, hardware requirements, and temporal resolution of output data in a extremely low-power FPGA target.
- The use of statistical analysis to calculate the maximum amount of channels that could be hosted on-implant within power budget limits, given the variable-codeword lengths.

The rest of this chapter is structured as follows. Section 4.2 describes the dataset used in this work and the different compression schemes. Section 4.3 shows the results of compression, decoding, and related hardware power consumption and resource usage. A recommended setting is then given that trades off among these metrics. Section 4.4 discusses some design consideration based on the results and Section 4.5 concludes this chapter.

4.2 Materials and Methods

The public datasets were loaded with Python 3.8 and MATLAB 2020a, the analysis was performed in Python 3.8, and the FPGA design in Modelsim Lattice Edition and iCEcude2 2020. The analysis Python code and FPGA Verilog code and designs have all been made publicly available at [103]. The formatted data and results have been made available at [104]. Researchers can use these to select their own compression system depending on their overall system requirements.

4.2.1 Datasets

To get a broad sample of MUA conditions, the Flint, Sabes Processed and Brochier datasets were used (Table 2.4). For each dataset, the SUA data was intra-channel collated to MUA, then binned to the desired BP. The behavioral data was resampled to the same BP resolution using linear interpolation.

For both the Flint and Sabes datasets, the BDP metric was as defined in Section 2.5.3. However, in the Brochier et al. dataset, the behavioural data consisted of labelled actions. As these were not continuous measurements, the behavioral decoding for the Brochier et al. dataset was not analysed in this work so as to keep the BDP metric consistent.

4.2.1.1 Training-testing data split

The data was split into training (A) and testing (B) data. This was because many different systems with different parameters were considered, e.g. different BPs and S values, different module combinations, etc. Therefore, the training data was used to identify a well-performing system. The final system was then tested on the test data B so as to give an unbiased estimate of the system performance on new data.

The Flint dataset was split so that the first 4 days of recording sessions were included in A. This corresponded to 10 out 12 recording sessions. The remaining 2, taking place over another day, were used as testing data and included in set B. The Brochier dataset was all included in the testing data B. Finally, the Sabes dataset was split so that data from subject Indy was included in A, and the data from subject Loco was included in B. This was done so that the testing data included data from completely new subjects. This strengthened the test data, allowing us to test the system on new subjects to see if the BR performance and BDP were as desired.

4.2.2 Compression Modules

The full system overview is given in Fig. 4.3. Different module combinations (Detailed in Section. 4.2.2.5) were investigated and a full grid search of all system parameters was performed. We investigated each system in terms of BDP, temporal resolution, hardware resources and on-implant power consumption. That allowed us to analyse and trade-off among different metrics of interest for a WI-BMI so as to identify the best configuration. Finally, we tested the selected configuration on neural data from new subjects, and confirmed its compression and behavioral decoding performance. Details of the modules are given below.

4.2.2.1 Binning and saturation

Two lossy compression steps have been applied to compress the MUA data. Binning the MUA data at a certain BP to obtain the FR is a primary means of compressing the MUA data. We also investigated saturating the FR at a maximum value $S-1$ to limit its dynamic range, where all FRs $> (S-1)$ were set to $(S-1)$. This means that there are fewer FR values that can be communicated, reducing the communication bandwidth. In order to test how different BP and S values can affect the compression and decoding performance, BPs of $\{1, 5, 10, 20, 50, 100\}$ ms and S values of $\{3, 5, 7, 9\}$ were tested.

These two operations perform a lossy compression to MUA signal, and therefore the degree to which they degrade the BDP was evaluated. The method is described in Section. 4.2.3.

4.2.2.2 SH encoding

Applying SH encoders can losslessly compress the MUA data. As in Table 2.2 (a-b), the idea is to give shorter codewords to more common values in an extremely hardware-efficient way. In this case, we give shorter codewords to more common FRs. The SH encoders were of length S, i.e. they had S input values and output codewords, representing FRs between 0 and $S-1$.

SH encoders need to be trained before use on representative data. Based on our observation on various recordings, we found that the firing rate distribution on average followed a decaying exponential, where smaller FRs were more common than larger FRs. This is shown in Fig. 4.1. As such, we trained the SH encoders on a decaying exponential, so they gave shorter codewords to smaller FRs. Further details are given in Appendix B.1. The SH encoder is represented by the 'Encoder(s)' block in Fig. 4.2. The use of multiple SH encoders will be discussed soon in Section 4.2.2.4.

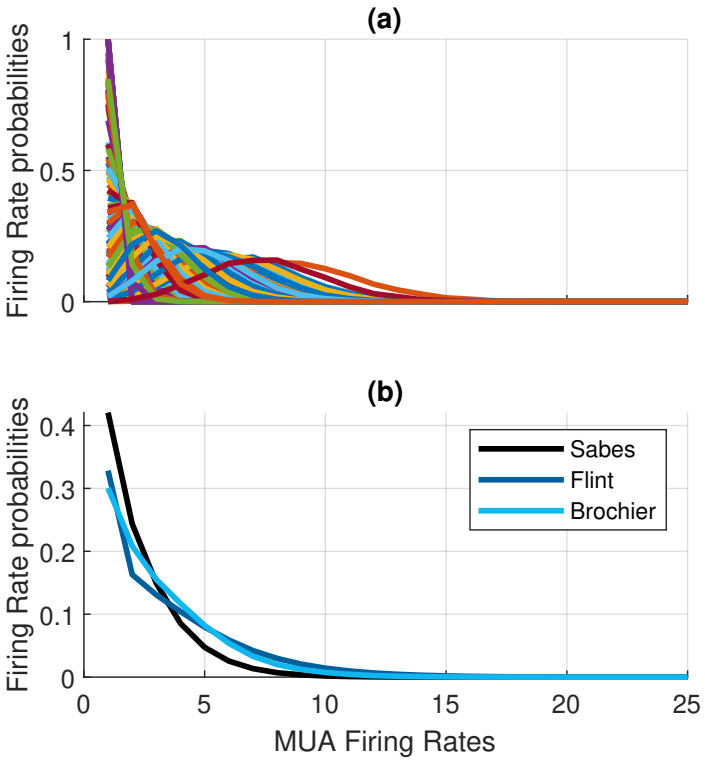


Figure 4.1: (a) A random sample of 100 channels' MUA FR probability distributions with a 100 ms BP. (b) Average MUA FR probability distribution for each analysed dataset, with a 100 ms BP.

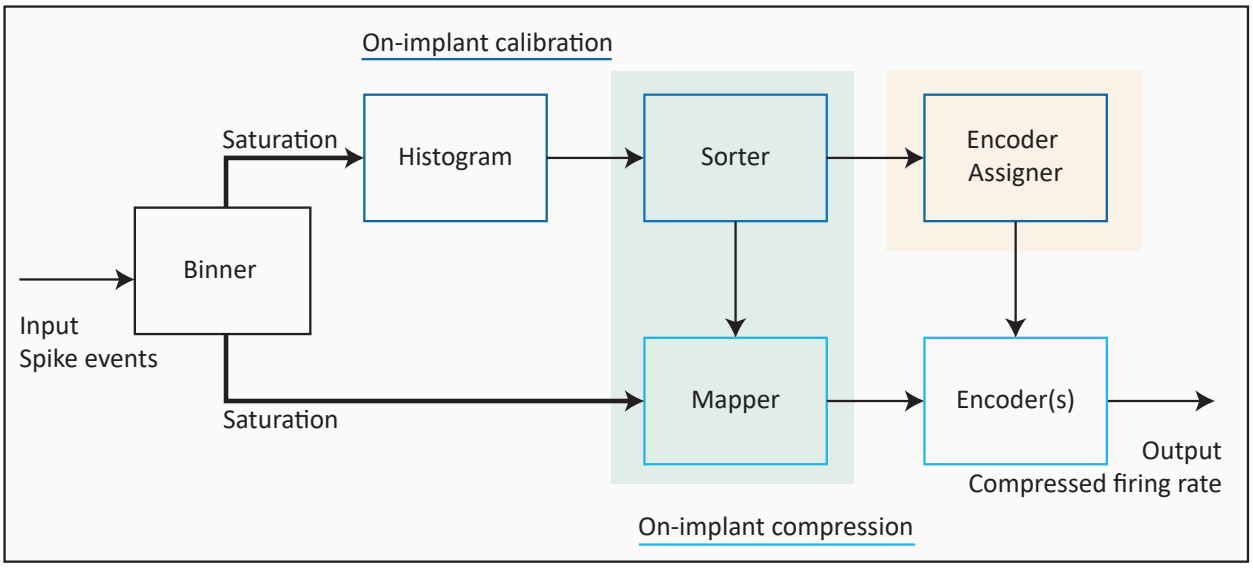


Figure 4.2: Compression data flow for each configuration. In the full system, a portion of the data is used for on-implant calibration, i.e. used to train a sample histogram for mapping and encoder selection. The mapping and selected encoders are then transferred to the main compression data flow, where the rest of the data is compressed. In the version without sorting and mapping, i.e. the 'Without Mapping' configuration, the green shaded modules are removed. If only $u = 1$ Huffman encoder is used on-implant, then the orange shaded module is removed, as no assignment is necessary. Finally, in the 'Only Binning' configuration, no on-implant calibration is performed, and the data goes straight from binner to transmission without a encoder.

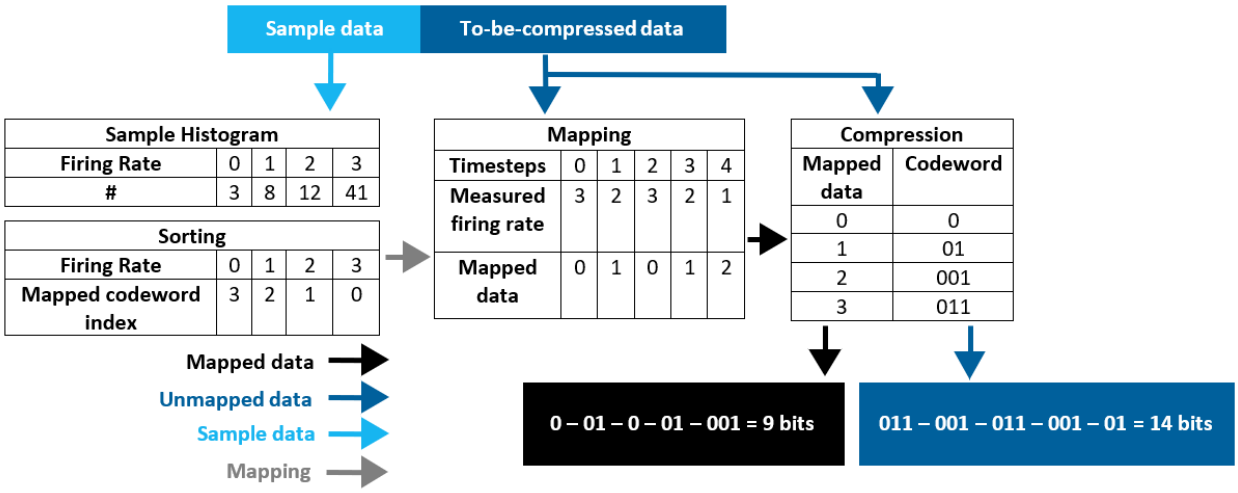


Figure 4.3: Use of a sample histogram to improve bit rates. A sample histogram is derived from the beginning of each channel's recording. It is then sorted using a hardware-efficient sorting, where smaller indices are given to more common firing rates. The sorting is stored as a mapping, used to sort the rest of the data, i.e. the to-be-compressed data. The data is then compressed after mapping, where if a firing rate was found to be the x^{th} most common in the sample histogram, it was given the x^{th} shortest codeword. As such, the data histogram is approximated by taking a sample, and if the sample is well-representative of the rest of the data, this may help shorter codewords be given to more common firing rates, improving compression. In the example we can see the mapped compressed data requires only 9 bits, relative to the unmapped compressed data which requires 14 bits.

4.2.2.3 Firing rate mapping using histogram

As shown in Fig. 4.1 (b), for BP $\leq$ 100 ms, smaller MUA FRs are on average more common than larger ones. However, as can be observed in Fig. 4.1 (a), this is not always the case for each individual channel. As such, assigning shorter codewords to smaller FRs will not always give optimal compression. Here we investigate the use of a sample histogram to address this problem. The beginning of each channel's recording was used to fill a sample histogram. This histogram was then used to estimate the relative frequencies of the FRs for each channel. The most common FRs in the histogram were then, for the rest of the data in each channel, assigned the shortest codewords via a hardware-efficient sorting (Appendices, Section B.4). This was referred to as mapping the most common FRs to the shortest codewords, given the sample histogram estimate. As such, some semi-adaptability was introduced into the SH encoders. A demo histogram sorting and mapping process is represented in Fig. 4.3.

This module is represented by the 'Histogram', 'Sorter' and 'Mapper' blocks in Fig. 4.2. We considered histogram sizes of $d = \{0, 2, 4, 6\}$ bits/bin, where there were S bins. Once 2^d samples had been measured, the histogram was considered to be full and was then used to estimate the FR frequencies. In the case of $d = 0$ bits, no histogram, sorting or mapping was used.

4.2.2.4 Utilising multiple SH encoders

Multiple SH encoders can be used to increase the on-implant compression adaptiveness. This works by using the sample histogram to estimate which encoder would compress each channel the best. Each channel is then assigned its optimal encoder. Such assignment was obtained by taking the dot product of the histogram and the Sorted Codeword Length Vector for each encoder. Dividing the dot product by BP and 2^d gives the BR.

As such, we assigned each channel to the encoder that gave the channel histogram the smallest dot product (i.e. BR).

To give more information, the Sorted Codeword Length Vector is a vector of integers that represent the length of each of the SH codewords. For example a SH encoder of

$$\{0, 10, 110, 111\}$$

would have an Sorted Codeword Length Vector of

$$\{1, 2, 3, 3\}$$

The dot product gives the total size of that channel's communicated data after compression using an equivalent SH encoder.

For a SH encoder of size S, there are h possible non-redundant (i.e. with unique Sorted Codeword Length Vectors) SH encoders. We designed a custom machine learning offline algorithm to select the top co-performing u encoders from amongst all h possible encoders. In other words, this selected the best combination of u SH encoders by using offline MUA training data. This ensured that we had the best u encoders on-implant that channels could be assigned to, with $u \in \mathbb{Z}^+$ and $1 \leq u \leq h$. The multiple on-implant SH encoders are represented by the 'Encoder Assigner' and 'Encoder(s)' block in Fig. 4.2. If $u = 1$ there was only one encoder on-implant, and so assignment was redundant since all channels went to the same encoder. We considered u values of {1, 2, 3, 5, 7, 10, 15, 20}.

The machine learning algorithm is further detailed in Section B.1.3 of the Appendices. It is an offline algorithm used in selecting which u SH encoders go on-implant, and is not itself present on-implant.

4.2.2.5 Module combinations

The modules included in each system configuration in Fig. 4.2 are given in Table 4.1. Similarly, the total parameter space investigated for the data compression is given in Table. 4.2.

There are in total 5 combinations. The first is the simplest system with only the binner and saturation, referred to as the 'Only Binning' system. The others use Huffman encoding, where the binned and saturated data goes straight to SH encoding. If multiple encoders are implemented, histogram and encoder assignment modules are needed in a calibration phase to select the best encoder for each channel. If we assume the FR data is not distributed according to a decaying exponential, the sorter and mapper are needed to map the more common FRs to shorter codewords. Therefore, the remaining four combinations are according to w/ or w/o mapping and $u = 1$ or $u > 1$. We refer to the one combination using all modules as the 'Full System'. The FPGA implementation of the different modules is included in Appendices, Section B.4.

4.2.2.6 Communication power estimation

For each parameter combination in Table. 4.2, we measured the compressed data BR using the training data A. From the BR, we derived the communication power from Eq. 2.1.

4.2.3 Impact of Lossy Compression on Behavioural Decoding Performance

Lossy compression involves losing information. In this case, the lossy aspects are increasing the BP, which reduces the temporal resolution of the neural data, and decreasing S, which saturates the data at an FR of $S - 1$. It is important to ensure that the lossy compression does not lose key information needed for the final application. In this case, the final application is the behavioral decoding of hand kinematics, which is a standard BMI behavioral measure.

Table 4.1: Required modules for each system configuration. The histogram is used for both assignment of encoders to channels, and for sorting/mapping of FRs. The system configurations vary if assignment is required or not, e.g. if only 1 encoder is considered, or if the histogram is to be sorted or not. The configuration also varies if no Huffman compression is considered, in which case only a binner and saturation are required.

	With Huffman encoding			
	With Mapping		**Without Mapping**	
	$u > 1$	$u = 1$	$u > 1$	$u = 1$
Binner and Saturater	✓	✓	✓	✓
Histogram	✓	✓	✓	✗
Encoder Assignment	✓	✗	✓	✗
Encoder(s)	✓	✓	✓	✓
Sorting and Mapping	✓	✓	✗	✗

Without Huffman encoding / Only Binning	
Binner and Saturater	✓
Histogram	✗
Encoder Assignment	✗
Encoder(s)	✗
Sorting and Mapping	✗

Table 4.2: Considered subset of analysed parameter space.

Subset of parameter space	
BP (ms)	1, 5, 10, 20, 50, 100
S	3, 5, 7, 9
d (bits per bin)	0, 2, 4, 6
u	1, 2, 3, 5, 7, 10, 15, 20

As such, to ensure that not too much relevant information is lost, a behavioral decoder was implemented. It was used to decode the hand X and Y-axis velocities, using the BDP metric from Eq. 2.3. The input to the decoder was the binned and saturated neural data, for BP values of {1, 5, 10, 20, 50, 100} ms. To be exhaustive in our behavioral analysis, S values from 2 to 59 were investigated for their effect on BDP. However, due to the impossibility of automating all of the hardware optimisation, only S values of {3, 5, 7, 9} were investigated for the compression work.

A Wiener Cascade Filter was used for the decoder. Wiener Cascade Filters have been found to have good decoding neural performance relative to other simple decoders, although they have generally found to not be as effective as deep learning methods [16, 91, 105]. However, their training times are significantly shorter [105]. As such, in this work they were used to investigate the relationship between S, BP and BDP. In this work, the Wiener Cascade Filter code from [16] was used. 5-fold cross-validation (hyper-)parameter optimisation was performed, and the details given in Appendix B.2. Once the parameters were optimised for each S and BP, the BDP was calculated for each combination using separate testing data.

4.3 Results

4.3.1 The impact of lossy compression

4.3.1.1 The impact of lossy compression on BR

It was proposed in [74,102] that increasing BP would lossily compress MUA data, but neither evaluated how efficient that compression is. To the best of the author's knowledge, the amount of compression to MUA from increasing BP is empirically shown for the first time in this work. The BR required to communicate a channel of binned data is equal to m/BP (bps/channel), where m is the number of bits required to represent the unsaturated dynamic range (S') of the FR. There is an approximately linear relationship between the lossless FR dynamic range and BP (Fig. 4.4 (a)). This produces a positive, approximately logarithmic effect on m from increasing BP (Fig. 4.4 (b). As such, merely increasing BP decreases the communication bandwidth (m/BP), relative to a lower BP (Fig. 4.4 (c)).

Saturating the FR range by setting the dynamic range S to $S < S'$ can obviously further reduce the BR, as the BR is proportional to the dynamic range to be transmitted. That is not the only advantage of saturating. If a SH encoder is used, a large glossary of the possible FRs to be compressed means a large SH codebook. Limiting the max FR from 10s to less than 10 can significantly reduce the size of the on-implant SH encoder and therefore reduces the power and resources.

4.3.1.2 Impact of lossy compression on BDP

Fig. 4.5 (a) shows the BDP vs. BP and S results averaged across the Flint and Sabes datasets. Examples of observed and predicted behavioral data are shown in Fig. 4.5 (b-d). Detailed results are given in Appendix B.3.

Fig. 4.5 (a) shows that the BDP improves as a function of BP, and is unaffected by S if S is large enough. For BP $\leq$ 20 ms, even $S = 2$, i.e. binary representation, is not lossy enough to affect the BDP. For BP at 50 or 100 ms, $S = 3$ or 5 are large enough to have less than 2% BDP degradation. For example, a 100 ms BP and an S value of 5 would give a BR of $ceiling(log_2(S'))$/BP = 30 bps/channel, while using no lossless compression. A typical bit rate of MUA signal is 1 kbps. Therefore, a 33 times bandwidth reduction can be easily achieved with just the use of binning and saturation lossy compression, with minor to no degradation on BDP, although the BTR is significantly affected.

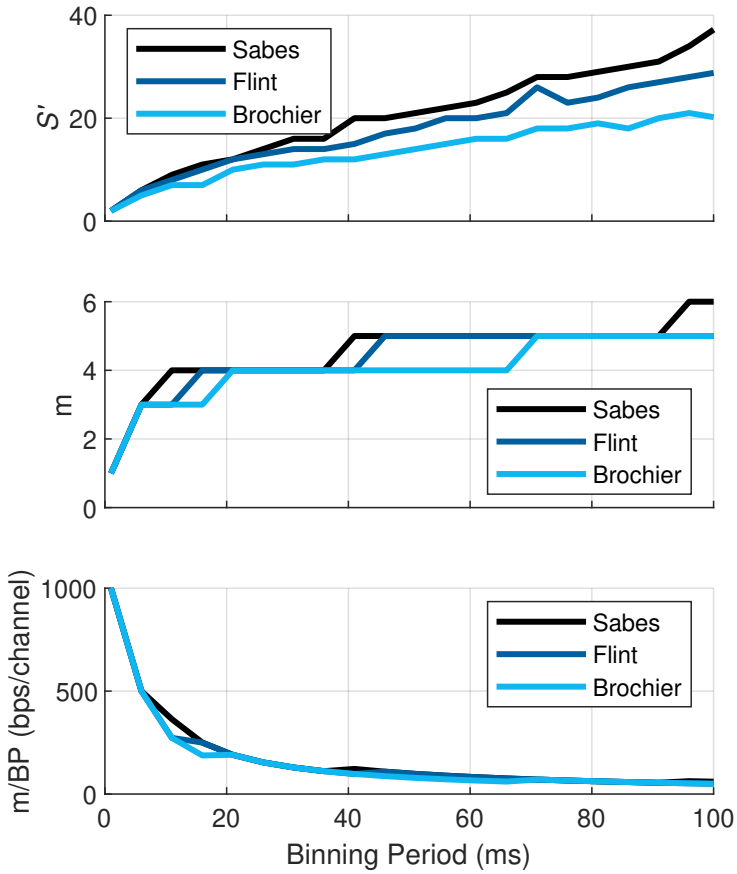


Figure 4.4: (a) A plot of $S' = max(X)+1$, the dynamic range of the MUA FRs, as a function of BP, where X is the multi-channel MUA data. (b) The number of bits $m = ceiling(log_2(S'))$ required to losslessly represent the dynamic range S' as a function of BP, without any lossless compression. (c) The communication bitrate m/BP (bps/channel) required to communicate a dynamic range of S'. (a-c) The analysed MUA data X, from which S' is measured, is the entirety of the training data A.

4.3.2 The impact of Static Huffman encoding

The SH encoder is a lossless compression operation applied after binning and saturating. Fig. 4.6 shows the reducing effect on BR from the addition of just a single SH encoder.

Overall, roughly another 50% bandwidth reduction can be achieved by using a single SH encoder. Moreover, as benefited from the saturation, the SH encoder with a small codebook can be implemented with only minor resources (less than 100 logic cells) consuming negligible power compared to the binner.

4.3.3 The impact of improving adaptiveness

Using an on-implant histogram to map FRs and select a suitable encoder from multiple on-implant encoders can improve the adaptiveness of the compression. This can be especially effective when the distribution of FRs to be compressed is unlike in the training data.

The results show that using a histogram to map FR can reduce the BR by another 5% to 20% as the histogram size increases. This improvement is only noticeable when the BP is 50 ms or 100 ms because longer BPs cause the FR distribution to vary more from the standard decaying exponential. In which case, increasing the histogram size estimates the true distribution more accurately, and so provides a more

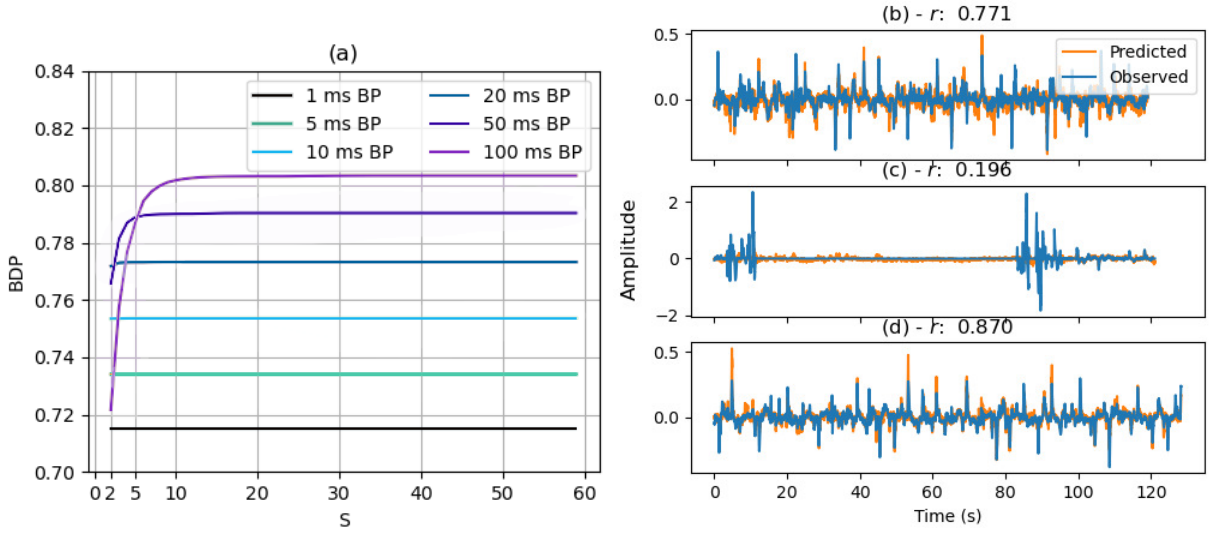


Figure 4.5: (a) Behavioral decoding performance (BDP) as a function of BP and S. Each S/BP combination was parameter optimised on 5-fold CV, with the results averaged from the Sabes lab and Flint datasets. (b-d) Example observed vs. predicted X-axis velocities from 5-fold CV, with corresponding BDP (r) for random Flint recording and parameter combinations during parameter optimisation at a BP of 5 ms.

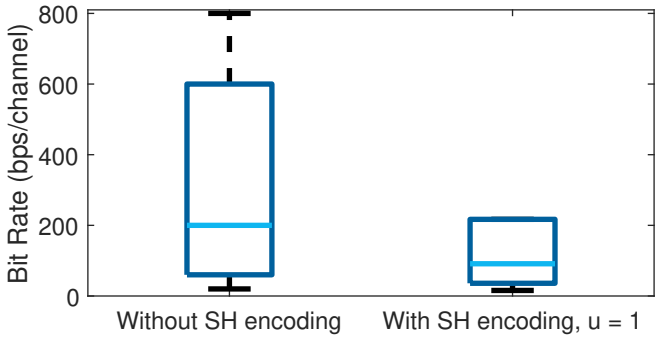


Figure 4.6: Boxplot of bit rates of all compression systems with and without Huffman encoding at u=1. This shows that the addition of a single SH encoder improves compression performance by more than 2 times on average.

accurate mapping, making the effect of the histogram notable. However, the FR distribution of short BPs rarely deviates from the decaying exponential, which makes the mapping mostly redundant. Mapping the FR with local information can even degrade the compression performance, regardless of BP, when the histogram size is too small. This is because if the beginning of the recording is not representative of the rest of it, the mapping may be maladaptive and perform worse than the assumed decaying exponential. As such, larger histogram sizes are more reliable.

Using more encoders does not significantly improve the BR. Observing the results showed that the same encoder was nearly always selected even when there were multiple available encoders. This 'best' encoder was the one trained on a sharp decaying exponential, where the Sorted Codeword Length Vector was of form $\{1\ ,\ 2,\ \ldots\ ,\ S-1,\ S-1\}$. That suggests, on the one hand, that the proposed machine learning-based encoder training algorithm works well in selecting the best encoder. On the other hand, the FR distribution when BP is less than 100 ms does not vary enough to require the adaptiveness provided by having more than one encoder.

Table 4.3: Chosen system parameters and encoder for testing and associated training (A) and testing (B) results. For the training and testing results, all data in A and B were respectively used.

Chosen parameters	
Architecture	Full system
BP (ms)	50
S	3
#Enc (u)	1
Histogram size (d)	6 bits/bin

Training Results (Flint, Sabes)		**Testing Results (Flint, Sabes, Brochier)**
Resources	246	246
BR (bps/chan)	26, 28	27, 28, 21
Dynamic Power (uW/chan)	1.49, 1.52	1.49, 1.52, 1.37
BDP	76%, 77% (2% and 1% reductions from S = max(x))	72%, 62%, null (1% and 0% reductions from S = max(x))

Encoder (Symbol → Codeword)
0 → 0; 1 → 01; 2 → 11

Furthermore, introducing adaptiveness comes with additional hardware costs. The on-implant histogram, dot product and sorting are all resource-hungry. Although intensive hardware optimisation was performed (details are provided in Supplementary Section B.4), their resource occupation can still be the bottleneck of the whole system.

4.3.4 System configuration selection

All the combinations of different modules, BP/BTR, S, histogram size and the number of encoders shown in Tables. 4.1 and 4.2 were produced and tested. The resulting BDPs, BTRs, hardware total power (communication + processing) and resource usages were then analysed together, as shown in Fig. 4.7 (a). Each point stands for one parameter setting and different colors represent the system operating at different BTR. The points high up on the resources axis indicate systems with larger S, histogram size or more on-implant encoders. The points that are perpendicular to the rest, with low resources but high power, are the setting with only the binner at different S values, as these had low hardware usage but higher communication power since no lossless compression was used.

4.3.4.1 The selection of BTR and S

Increased BTR can bring higher BDP and lower communication power as the FRs are transmitted less frequently. However, S needs to be sufficient for high BTRs to enable sufficient BDP. It was observed that at a BTR of 100 ms and $S = 3$ that the BDP suffered significantly. This can be observed in both Fig. 4.5 (a) and Fig. 4.7 (a) as a cluster of dark points (100 ms BTR) with low BDP. As such, given sufficient S, increasing the BTR is a very attractive prospect. It needs to be balanced with the desired temporal resolution of the decoded output, but it was decided that a 50 ms BTR was worthy of consideration for the test case.

Regarding behavioural decoding, 50 ms can be a good choice. It is tolerable in terms of the impact on user delay, assuming a smoothing function for the cursor, while also having high BDP and low communication power. As such, a BTR of 50 ms was selected for the test case. Furthermore, according to Fig. 4.5, when the BTR is 50 ms, limiting the FR dynamic range to 3 produces a 2% BDP degradation. As such a combination of BTR = 50 ms and $S = 3$ was selected.

4.3.4.2 The selection of the number of encoders and histogram size

As previously stated, using more than one encoder does not significantly reduce BR and therefore was not considered further. Using no SH encoder can be especially resource-saving. However, the approx. 50% bandwidth reduction brought by a single SH encoder can significantly reduce the communication power and therefore the total power. The points that belong to a perpendicular segment relative to the rest of the data in Fig. 4.7 show how the addition of SH can significantly reduce communication power.

With respect to the histogram size, although it needs more resources, it improves the compression while the resource usage is still acceptable. To get more insight, we focused in on Fig. 4.7 (a) with BTR = 50 ms, resources < 260 and dynamic power $< 2.2\,\mu$W/channel, shown in Fig. 4.7 (b). The bottom four points from left to right are the configurations with one encoder, $S = 3$, and histogram sizes of $\{0, 2, 4, 6\}$ bits/bin respectively. It makes sense, in terms of scaling with channel count, to prioritise the lowest power configuration. This is because resources are roughly static at 246 with increased channel count, which is acceptable. Additionally, BDP increases somewhat logarithmically with channel count according to neuron dropping curves [30, 106], and BTR is unaffected. However, power increases roughly linearly with channel count, making it the parameter that scales least well. Therefore as long as the resource usage is acceptable, reducing the power consumption should be the first priority. However, for resource-constrained scenarios, the bottom left setting can be selected which is the configuration without histogram, i.e. no mapping, where the encoder compresses the binned firing rate directly (for data flows see Fig. 4.2).

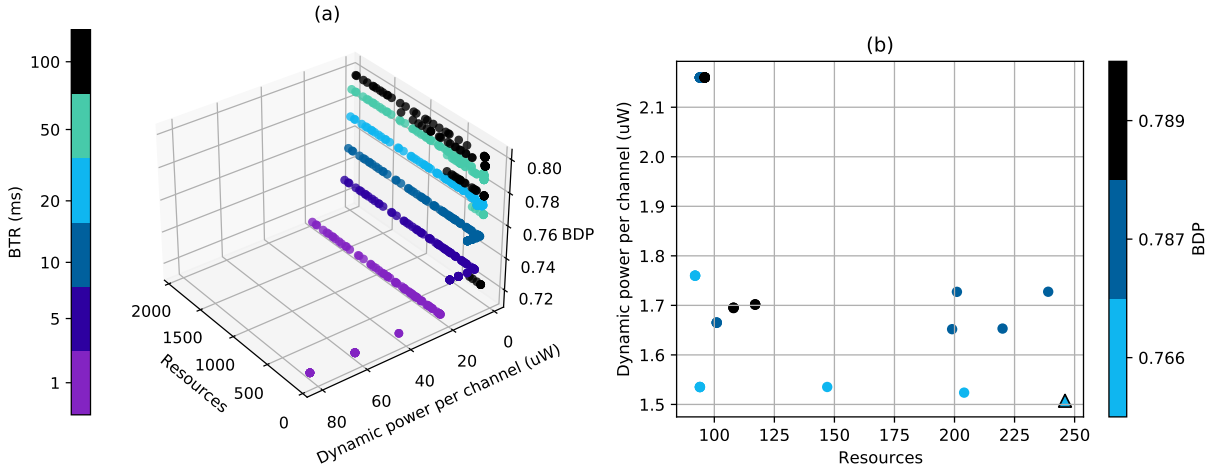


Figure 4.7: (a) Integrated results: BDP and BTR values for different resources and dynamic power consumption levels. (b) Sample of integrated results, with BP/BTR = 50 ms, resources < 260, dynamic power $< 2.2\,\mu$W/chan for our 128 channel system. The outlined triangle represents the chosen system configuration for our tested system. Note that the color bars for (a) and (b) are distinct.

4.3.5 Testing data results

Next, the chosen system was tested on data it had not seen yet. Given the chosen system parameters, summarised in Table 4.3, the BR and BDP were determined on the testing data B, also shown in Table 4.3.

4.3.5.1 Communication power results

For the Flint, Sabes and Brochier data in B, the across-channel-and-recording average BRs were 26.5, 27.8, and 20.6 bps/channel respectively, corresponding to dynamic power/chan values of 1.49, 1.52 and 1.37 μW. These are highly similar to those in the training data, where the averages for the Flint and Sabes data were

26.4 and 27.8 bps/chan and 1.49 and 1.52 μW respectively. This is especially significant for the Sabes and Brochier test data, which consisted of subjects that were completely separate from the data in A. This means that the system effectively compressed data from 3 entirely new non-human primate subjects. This can be compared to the 40 bps/channel produced by the Only Binning configuration where no Huffman encoder is used (ceiling($\log_2(3)$)/(50 $\times 10^{-3}$)).

4.3.5.2 Behavioral decoding results

The BDP was measured for the Sabes and Flint datasets using the testing data in B. Each channel was split 90-10% into training and testing sets. S and the BP were fixed at 3 at 50 ms respectively, and as in Section 4.2.3 the Wiener Cascade Filter hyper-parameters and pre-processing parameters were 5-fold cross-validated on the training set. The best parameters for each BP/S combination were taken, and the BDP measured on the testing set.

The average BDP for the Flint dataset was 0.724, and for the Sabes data it was 0.616. These BDP values from B are significantly lower than in the training data A. At first glance this is worrying, since it may suggest the compression scheme was overly lossy and too much behavioural information was lost. However, close examination of the results indicate that the tested system's BDP values are lower because the recordings have less behavioral information in them, or the recording quality is less good, etc. This is shown by comparison of Fig. B.5 and B.8 in the Appendices, as well as Fig. 4.5. In particular for the Sabes test data, they show that the data seemed to suffer from a few recordings with very low BDPs (e.g. at approx. 0.4), which dragged down the average to significantly below the median BDP. A larger amount of Sabes test recordings performed quite well (approx. 0.7), although they still generally performed worse than the train A data. As such, the test recordings seem to simply be of worse quality than the train recordings.

Furthermore, across all test recordings, the tested system's compression did not negatively affect the BDP by more than 1.62% compared to the top observed BDP for each recording across all BP and S, and for the overwhelming majority of recordings the impact was 0. This is encouraging, as it shows that the quality of the recordings was the decisive factor in the BDP, not the lossy compression.

The main takeaway for the BDP results is that reducing S to 3 for a BP of 50 ms had no significant negative effect on BDP. It is expected that if the recording quality is similar to that in the training data, higher BDPs will result. It is also likely that using more advanced deep learning decoders would result in higher BDPs [16, 91, 102].

4.4 Discussion

4.4.1 The number of channels supported by the power budget

A 1 mm $\times$ 1 mm scale FPGA implant, with a maximum permitted heat flux of 10 mW/cm$_2$, has a power budget B of 200 μW. This assumes equal heat flux from both faces of the implant, and negligible heat flux from the edges. With such a budget, one could wirelessly transmit up to 10 uncompressed MUA channels if all the on-implant power was used for communication and the communication energy was 20 nJ/bit. In practice, with a static FPGA power of 162 μW, negligible spike detection power [88] and a processing power for the 1 ms binner of 0.96 μW/channel, a maximum of 1 channel could be measured on-implant.

How many channels can our chosen compression system host while remaining within the power budget? The chosen system depends on variable-length codewords. Therefore, there is a risk that the BRs will be higher than the expected $\sim$27 bps/channel as given in Table 4.3. For example, one might measure a handful of particularly active channels, and this may increase the BR. If one chooses the number of channels on-

implant so as to be close to the permitted power budget, and the channels are more active than expected, this may produce more heat than desired. As such, it warrants choosing the number of channels based on a statistical understanding of the worst case scenarios.

As such, a random sample of the channels was selected. For sample size $z \in \mathbb{Z}^+$, 10,000 samplings were taken of z random channels. For each random sampling Y, from the channels' summed BRs we obtained the resulting total system power P using the estimates in Equation 4.3:

$$\begin{aligned} P_Y &= \sum_{i=1}^{z}(\mathrm{BR_i}) \times \text{Comm. Energy} + z \times \text{Processing Power} + \text{Static FPGA Power} \\ P_Y &= \sum_{i=1}^{z}(\mathrm{BR_i}) \times 20\,\mathrm{nJ/bit} + z \times 0.96\,\mu\mathrm{W/channel} + 162\,\mu\mathrm{W} \end{aligned} \quad [\mathrm{W}] \quad (4.3)$$

where $\mathrm{BR_i}$ is the BR of the $\mathrm{i^{th}}$ channel in sampling Y, where $1 \leq i \leq z, i \in \mathbb{Z}^+$.

It was then determined, for each number of channels z, what percentage of random channel combinations exceeded the desired power budget $B = 200\,\mu\mathrm{W}$:

$$p(z) = \frac{1}{10^4}\sum_{Y=1}^{10^4}(P_Y > B) \quad (4.4)$$

where $(P_Y > B)$ is a boolean value equal to 1 if $P_Y > B$ and 0 otherwise. As such, $p(z)$ gave a permutation derived p-value for each number of channels z not exceeding the power budget.

Using the chosen architecture and power budget of $B = 200\mu\mathrm{W}$ and averaged across the 30 CV runs, it was found that from the training data results that having up to 23 channels never exceeded the power budget. Having 24 channels had a p-value of ∼3e-3 of not exceeding the power budget, and 25 channels or higher had significant chances of exceeding the power budget of $p(z > 24) > 0.05$. As such, we think having approximately 22 channels for our 1 mm × 1 mm FPGA hardware is ideal assuming the given power estimates hold true, while staying within a conservative heating safety margin. As such, by compressing the MUA data one can send out over 22 times as many channels as when sending out the raw MUA data for a similarly sized FPGA device. In ASIC, this difference would likely be far more pronounced, given the reductions in dynamic and static power. However, the contribution of the front-end amplifier and ADC would need to be included as they would likely be integrated. Given that ADCs with power consumption as low as $0.87\mu\mathrm{W/channel}$ have been achieved [67], there is reason to believe that impressive channel counts could be obtained at mm-scale in ASIC.

Furthermore, if one increases the communication energy to 100 nJ/bit, not a single uncompressed channel could be communicated by a similarly sized FPGA device. However, 15 channels, compressed with the chosen system, could be communicated within the power budget.

4.4.2 Configuration selection considerations

If increased BDP is an absolute priority, then a configuration with a higher S may be appropriate. However, one should consider that if increased power is required as a result, this can reduce the amount of allowable channels for an implant of the same size, perhaps reducing the final BDP. Similarly, if a system has only a few recording channels, where communication power does not dominate, one may choose a system that significantly reduces hardware requirements over marginally reducing BR, e.g. a 'Without Mapping' configuration.

It may be that a system with a small BTR should be prioritised. Unfortunately, the grid search of the windowed SH method found an unfortunate trade-off between BTR and communication power. The systems

with good BTR tended to be relatively power-hungry. For example, a BR of 100 bps/channel is required for a BTR of 10 ms. This BR may be too high. As such, this shortcoming of the windowed method will be tackled in the next chapter, investigating other means of compressing the MUA signal while maintaining good BTR and low BR.

4.4.3 The effect of BP on BDP

It was found that BDP increases as a function of BP between 1 and 100 ms. Although this differs from some other results that used different decoders [102], it has also been theorised in the literature that one should expect BDP results to vary by decoded behavior and decoding algorithm [74]. [102] found that increasing BP reduced BDP, but it used LSTM neural network decoders, which are a form of deep decoder. It is unsurprising that a deep decoder that can exploit long-term temporal dependencies to find extra information in high-precision timing of neural firing rates compared to a simple Wiener Cascade Filter decoder. As such, the difference between the relationship between BP and BDP in Fig. 4.5 and [102] is not surprising.

4.4.4 Generalising the Results to Other Behaviors

How well lossy compression has performed depends on the final use of the data, and whether any key information has been lost. In this work, the final outcome was the decoded hand kinematics, which are a standard BMI behavioral measure [16, 64, 65, 77, 96, 102] and the most desired by patient surveys. Therefore, although the lossy aspect of the compression system is tailored to a specific task, it is a general task that is ubiquitous across BMI research. Additionally, the lossy aspect of the data compression scheme is very simple: increasing the BP, which is standard during BMI behavioral decoding [64, 74, 77, 86, 87], and saturating the MUA data, which had only a negligible effect on BDP for this task.

For this hand kinematic task, the system's BDP was tested on a completely new subject, 'Loco' of the Sabes dataset, and for all tested recordings the BDP was at most negligibly reduced by data compression (1.6% in the worst case, 0% in most cases). For the Flint test recordings (new recordings on the same Flint subject as in the training data), the BDP was similarly unaffected by lossy compression. As such, we can say that the performance of the compression system was robust across different subjects, which is a very significant result. The tested compression scheme generalised very well to 3/3 new subjects in terms of compression, and to 1/1 new subject in terms of behavioral decoding for hand kinematic tasks in WI-BMIs. However, we cannot say the same across tasks, for lack of information. It is our belief that the results will probably be consistent across tasks decoded from the motor cortex, but if not, then S can simply be increased or BP varied.

4.4.5 Fixed Length vs. Variable Length Codewords and Bit-Flip Errors

It warrants mentioning that lossless compression works by giving variable length codewords to symbols. Due to the multiplexed encoding of MUA, this makes losslessly compressed MUA data more vulnerable to bit flip errors making the multiplexed communicated data block undecodable. As such, some noisy channel encoding or decreasing the BP may be necessary, assuming the bit flip error rate is sufficient to warrant it. This would increase the BR marginally, and is discussed further in Appendix B.6.

4.5 Conclusion

In conclusion, the objective of this chapter was to reduce the MUA data bandwidth. We did so via a large grid search of various compression architectures, and qualified the results based on total power, required

hardware resources, BDP, and BTR. We eventually achieved nearly 40 times MUA bandwidth reduction from 1 kbps/channel to 27 bps/channel with 2% decoding degradation on training data, and less than 1% on testing data, with a BTR of 50 ms. Such a distinguishable achievement is made by a binner at 50 ms BP, a dynamic range limited to 3 possible values, hardware efficient mapping using a histogram size of 6 bits and losslessly compressing the resulting signal with a pre-trained static Huffman encoder. Our results have been across validated using 3 datasets (Flint, Processed Sabes and Brochier) and 3 new subjects suggesting consistent compression performance. The system has been implemented on a FPGA platform using 246 logic cells, consuming only 0.96 μW/channel and can accommodate more than 300 channels within 4 kB RAM. All results and hardware designs are made publicly available, and researchers are free to select from them for their own system designs.

However, the compression at fine BTR was not satisfactory. For example, no compression was achieved at a BTR of 1 ms. Therefore, the next chapter will look at event-driven compression methods for the MUA signal, aiming to improve compression at fine temporal resolutions.

Author Contributions

The work in this chapter was done in collaboration with Zheng Zhang (Z.Z.). Compression and behavioral decoding work in this chapter was carried out by O.W.S., and hardware design, optimisation and processing power estimation work was done by Z.Z. Details on hardware design and optimisation and power estimation are given in Appendix B.

Chapter 5

Comparison to Event-Driven Architectures for Compressing MUA

This chapter addresses the weaknesses of the windowed MUA compression method at finer BTRs discovered in the previous chapter. It proposes a number of event-driven MUA compression schemes, and analyses their performance in terms of BR, BDP, hardware resources and total on-implant power. This chapter is based on work done in collaboration with Z.Z. He contributed the hardware designs and processing power estimations.

This chapter has been adapted from the following published article: OW Savolainen, Z Zhang, TG Constandinou, "Ultra Low Power, Event-Driven Data Compression of Multi-Unit Activity", bioRxiv, 2022

5.1 Background

A hardware-efficient MUA compression system was introduced in the previous chapter. At its base, it used binning, limiting the dynamic range and SH encoding. Additional techniques were also considered, such as the use of sample histograms for SH mapping and the use of multiple SH encoders for channel assignment. It achieved state-of-the-art compression performance, especially when the BTR was 50 ms or above. However, when the BTR was smaller, the compressed bandwidth was unsatisfactory. Given that there is a desire for finer temporal resolution in BMIs, to enable a faster control loop and improve user experience, some form of MUA compression, tailored to smaller BPs, is desired.

Shorter BPs make the firing rates sparse. In other words, in a smaller time window it is more likely that a FR of 0 will occur. Using this known property of MUA signals can help us improve the compression performance for BPs below 50 ms.

In this chapter, we propose several event-driven architectures, where only data from active channels is transmitted for the given BP, taking advantage of sparse MUA FRs at lower BPs. These are compared to the windowed method from the previous chapter. Overall, four compression schemes will be examined: the windowed scheme from Chapter 4, an Explicit Event-Driven (EED) method, which transmits the active channel IDs with corresponding spike counts, a Delta Event-Driven (DED) method, which transmits the delta-sampled active channel IDs with corresponding spike counts, and a Group Event-Driven (GED) method which uses a form of run-length encoding.

This chapter contains multiple original contributions to MUA compression. These are as follows:

- Design and analysis of novel MUA data compression architectures, such as the EED, DED, and GED encodings.
- The hardware designs, made publicly available, of these encodings.
- Comparison of these encodings to the state of the art in MUA compression. Very large improvements were made in terms of communication bandwidth for BTRs of 20 ms and below, appropriate for higher temporal resolution MUA-based BMIs. Compression performance was improved by up to over an order of magnitude for finer BTRs relative to the state of the art. This allows for significantly more MUA channels to be fit on-implant while staying within heating limits. This was found to be true using data from 3 different publicly available datasets, including 5 non-human primate subjects and a total of 23 hours of Utah array 96-channel recordings.
- We further strengthen the case for SH encoders. We show that SH encoders, trained on distributions that make simple assumptions about MUA data, perform just as well as non-causal Adaptive Huffman encoders while being far more hardware-efficient and causal. As such, SH encoders seem to be a very attractive means of data compression for MUA-based WI-BMIs.
- We investigate the use of sample histograms for mapping, as in [59], for the event-driven schemes proposed in this work. We determined their effect on BR and processing power, as well as analyzing trade-offs in their use.

The rest of this chapter is organised as follows: Section 5.2 introduces three datasets used in this work and details the compression methods. Section 5.3 shows the compression performance and hardware cost of each method. Based on these results, we give the recommend compression settings at different BP and channel counts, trading off between the BR, processing power and hardware cost. Section 5.4 discusses some algorithm and hardware design considerations in MUA compression and Section 5.5 concludes this work.

5.2 Methods

All compression algorithm work was done in MATLAB R2021A. All hardware design and optimisation work was done using ModelSim Lattice Edition and IceCube2020.12. All code and results are publicly available at [107].

For each encoding, the BRs were calculated based on the probability of each symbol occurring. All of the mathematical details are given in Appendix C.

5.2.1 Dataset and Data Formatting

Brain signal conditions can vary across subjects and tasks. In order to reduced the bias, we used three different publicly available datasets. As in the previous chapter, we used the Flint, Processed Sabes and Brochier datasets [64, 65, 96], summarised in Table. 2.4. For each dataset, the SUA data was intra-channel collated to MUA, then binned to the desired BP, where BP $\in \{1, 5, 10, 20, 50, 100\}$ ms. Based on the results in Fig. 4.5 from Chapter 4, we limited S to [2, 2, 2, 2, 3, 5] for BPs of $\{1, 5, 10, 20, 50, 100\}$ ms respectively.

The length of the recordings was largely irrelevant for the sake of this work, as all encoding was done without using a time derivative and the bandwidth was measured in [bps/channel], normalising for time. As such, a standard length of 100 s was set for each recording, long enough to gather a stable distribution for each channel for any of the tested BPs. To maximize the use of the data in observing the effect of channel count,

recordings were split into consecutive 100 s segments and collated together. For example, a 400 s recording was represented as 4 parallel channels of 100 s long recordings, where x_o,...,$x_{N/4-1}$ became one channel, $x_{N/4}$,...,$x_{N/2-1}$ became the next, etc., where x_k is a single-channel MUA recording and $N = 400$ s/BP is the length of the recording in samples. A total of 79200 channels were available after splitting and collation.

The data was then split into training and testing sets. This is because the encodings have parameters that need to be optimised, which was done on the training set. The testing of parameter-optimised encoding on the testing set can then provide an objective estimation of the encoding performance on the unseen data. The training-testing split was done by randomly selecting 30000 channels and placing them into the training set. The remaining 49200 were put into the testing set. For each encoding, BP, S and n combination, the training and testing results were each averaged across 5 training and testing runs. For each run, $n \in \{10, 100, 1000, 10000, 30000\}$ channels were selected randomly without replacement from amongst all training or testing channels. Different numbers of channels n were considered because the event-driven encodings performance depend son the number of channels.

5.2.2 Windowed encoding

The windowed encoding from the previous chapter was the first scheme to be implemented. As aforementioned, it is called "windowed" because it sends out the number of MUA events in a non-overlapping window of length BP for each channel, regardless of whether an event occurs on a channel or not. See Section 4.1.1 for more details. In this chapter, the windowed encoding was implemented using a single SH encoder, given the result from the previous chapter that 1 SH encoder was sufficient for compression while being significantly simpler to implement in hardware.

The SH encoder was pre-trained on a decaying exponential which mimics the distribution of MUA FRs at BPs ≤ 100 ms (Fig. 4.1), where smaller codewords are given to smaller FRs.

5.2.3 Explicit Event-Driven encoding

In the windowed architecture, the FR of each channel is encoded. The channel ID is implicitly encoded in bit position. To the best of the authors' knowledge, the following event-driven architectures are proposed for the first time in MUA compression. In these event-driven architectures, the channel ID is explicitly encoded and only active channels will be transmitted. i.e., the channel ID is only sent out if a non-zero FR occurs on that channel, followed by a binary codeword representing the rate. If a channel has a FR of 0, nothing gets communicated for that channel, and the offline decoder assumes the missing channels had FRs of 0. For sparse signals, i.e. where FRs not equal to 0 are rare, this can offer reduced bandwidth over the windowed paradigm, where even FRs of 0 need to be communicated for each channel.

As the channel IDs need to be explicitly encoded, the simplest method is to give the channels a standard binary codeword of length k_1, where:

$$k_1 = ceil(log_2(n)) \quad \text{[bits]} \quad (5.1)$$

Similarly, the MUA FR for each channel is encoded as a binary codeword of length m_2:

$$m_2 = ceil(log_2(S-1)) \quad \text{[bits]} \quad (5.2)$$

An exception occurs if $S = 2$, i.e. $m_2 = 1$ and i is limited to 0 and 1. In that case the FR codeword is unnecessary as the decoder can assume that all received channels have an FR of 1. An example of the encoding without lossless compression, with $n = 4$, $k_1 = 2$ and $m_2 = 2$, is given by:

0001 0100 1011

This shows that channel 1 (00) had a FR of 2 events (01) in the bin, channel 2 (01) had FR = 1 (00), channel 3 (10) had FR = 4 (11), and channel 4 had a FR = 0 (absent). In this work, this encoding was named the explicit event-driven (EED) encoding because the FR per channel is explicitly encoded in the m_2-length codeword that follows the channel ID.

To include the SH encoder, there was no good way to compress the channel IDs, since they are *a priori* equally likely to be active. Therefore, the EED encoding uses the same k_1 length codeword for the channel IDs, but uses varying length SH codewords for the FRs. As in the SH windowed implementation, the FR SH encoder was trained on a decaying exponential.

5.2.4 Delta Event-Driven encoding

The previous encoding suffers since the probability of each channel being active is *a priori* equal. As such, the channel IDs, in the form that they are, cannot be efficiently compressed with SH encoders. However, pre-processing can fix this problem. The channel difference between two successive active channels has varying probability, and therefore entropy encoding can be effective. In other words, we can encode the delta-sampled channel IDs of active channels. Therefore, in this 'delta event-driven' (DED) encoding, if a channel has a FR above 0, it is given a Δ value by subtracting its ID, $j_{current}$, from the ID of the previous channel to have a FR above 0, $j_{previous}$. I.e., $\Delta = j_{current} - j_{previous}$.

Then, the Δ-sampled channel IDs were compressed using a SH encoder trained on a decaying exponential. Significant memory optimisation was also done by setting a maximum Δ-sampled SH encoder size, using a form of run-length encoding. This reduced BR slightly but significantly reduced memory requirements. The details of this optimisation are extensively detailed in Appendix C.3 and C.4. The FRs were compressed with the same SH encoder as in the EED encoding.

5.2.5 Group Event-Driven encoding

In the GED encoding, one uses position and stop symbols to encode the FRs. As in the EED encoding, one explicitly encodes the channel ID, but here one encodes the FR per channel implicitly in channel ID position. For example, in decimal,

$$2\,4\,stop_2\,1\,6\,stop_3\,stop_4\,3$$

signifies that channels 2 and 4 had a FR of $i = 1$ in the given bin, channels 1 and 6 had a FR of $i = 2$, channel 3 had FR $i = 4$, and the rest of the channels had FR $i = 0$. For the codeword lengths, the simplest implementation is to give the stop symbols and the channel IDs a codeword of length k_2 bits from the same dictionary, where:

$$k_2 = ceil(log_2(n + S - 2)) \quad \text{[bits]} \quad (5.3)$$

E.g. $n = 2$, $S = 4$, means that there are 2 channels, FRs between 0 and 3 inclusive can be encoded, and $k_2 = 2$. Channel 1 gets a codeword of 00, channel 2 gets a codeword 01, and the stop symbols for $i = 2$ and $i = 3$ get codewords of 10 and 11 respectively. It essentially involves sorting the channels by FR, and using a form of run-length encoding for the channel IDs.

There was no clear role for SH encoding, since the stop symbols and FR codewords need to come from the same encoder. Estimating these probabilities *a priori* is difficult. As such, no SH encoding for the group encoding was used.

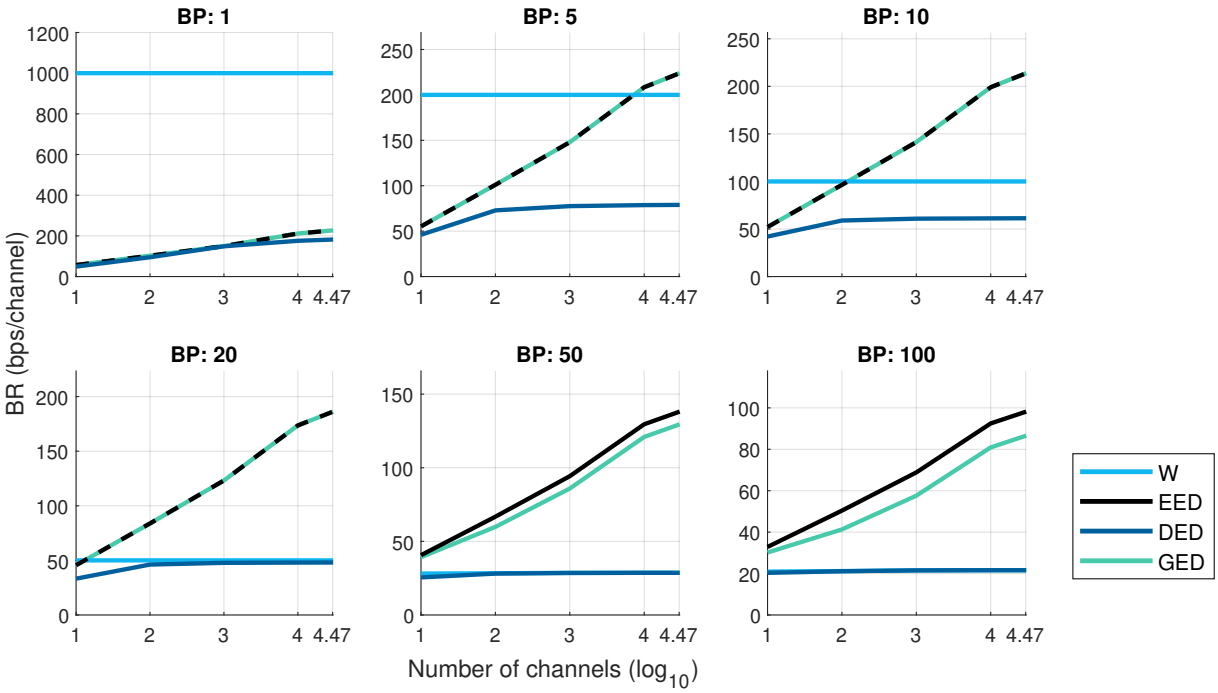


Figure 5.1: BRs for communication schemes at a BP $\in$ [1, 5, 10, 20, 50, 100] ms. For each BP, S was respectively fixed as [2, 2, 2, 2, 3, 5]. At $S = 2$, the explicit and GED encodings are mathematically identical, and so their BRs overlap for BPs $\leq$ 20 ms. 'W' stands for the windowed encoding.

5.2.6 Hardware Implementation

As in the previous chapter, the above encoding algorithms were implemented on the Lattice ice40LP1K FPGA in order to assess their hardware complexity (power consumption and resources usage). The hardware implementations are given in Appendix C.3. The hardware static power is 162 μW.

5.3 Results

In Fig. 5.1, we plot the BRs of each encoding at different BPs and n. We can observe that the windowed and DED encodings perform best, with the best one depending on BP and n. From the hardware perspective, the windowed encoding is far more hardware efficient than the DED encoding. Whilst the amount of logic cells is similar for the two encodings, the DED encoding uses significantly more memory, and therefore the processing power is higher. Additionally, the windowed scheme performs the same independent of channel count. However, the DED scheme is affected by channel count. When n increases, the delta-sampled channel IDs can be larger, meaning longer codewords and higher BRs. As such, increasing n slightly increase the DED BR, but significantly less than for the EED and GED encodings.

As such, the optimal selection generally varies as a function of channel count and BP. Ultimately, we want the total power on-implant to be reduced. As such, Fig. 5.2 shows the total dynamic power for the FPGA implementation, made up of the processing power for each encoding and the communication power, estimated as the BR multiplied by an estimated 20 nJ/bit communication energy.

Table 5.1: Suggested static encodings and corresponding BRs, hardware resources and dynamic power for each BTR and channel count. The BR and power results are from the test data. 'W' represents the windowed encoding. 'x' entries are where the memory requirements exceeded those of the FPGA platform we selected. The FPGA Logic Cells and Memory are shared across all channels.

(a) Recommended Encoding

		n				
BTR	**S**	**10**	**10^2**	**10^3**	**10^4**	**3×10^4**
1	**2**	DED	DED	DED	DED	DED
5	**2**	DED	DED	DED	DED	DED
10	**2**	DED	DED	DED	DED	DED
20	**2**	DED	W	W	W	W
50	**3**	DED	W	W	W	W
100	**5**	W	W	W	W	W

(b) Bit Rates (bps / channel)

		n				
BTR	**S**	**10**	**10^2**	**10^3**	**10^4**	**3×10^4**
1	**2**	50	99	151	177	183
5	**2**	45	75	78	79	79
10	**2**	44	62	62	62	62
20	**2**	31	50	50	50	50
50	**3**	25	29	29	29	29
100	**5**	20	21	22	22	22

(c) FPGA Logic Cells

		n				
BTR	**S**	**10**	**10^2**	**10^3**	**10^4**	**3×10^4**
1	**2**	114	144	164	207	242
5	**2**	124	153	174	217	252
10	**2**	130	159	180	223	258
20	**2**	135	128	134	162	192
50	**3**	183	171	176	207	x
100	**5**	232	246	251	286	x

(d) FPGA Memory (bits)

		n				
BTR	**S**	**10**	**10^2**	**10^3**	**10^4**	**3×10^4**
1	**2**	60	648	2448	20448	60448
5	**2**	60	968	2768	20768	60768
10	**2**	60	968	2768	20768	60768
20	**2**	60	200	2000	20000	60000
50	**3**	140	400	4000	40000	x
100	**5**	60	600	6000	60000	x

(e) FPGA Dynamic Power (μW / channel)

		n				
BTR	**S**	**10**	**10^2**	**10^3**	**10^4**	**3×10^4**
1	**2**	2.03	3.05	4.1	4.74	5.39
5	**2**	1.93	2.58	2.66	2.76	3.34
10	**2**	1.95	2.32	2.35	2.42	2.97
20	**2**	1.7	1.96	1.98	2.03	2.55
50	**3**	1.59	1.54	1.58	2.07	x
100	**5**	1.4	1.43	1.46	2	x

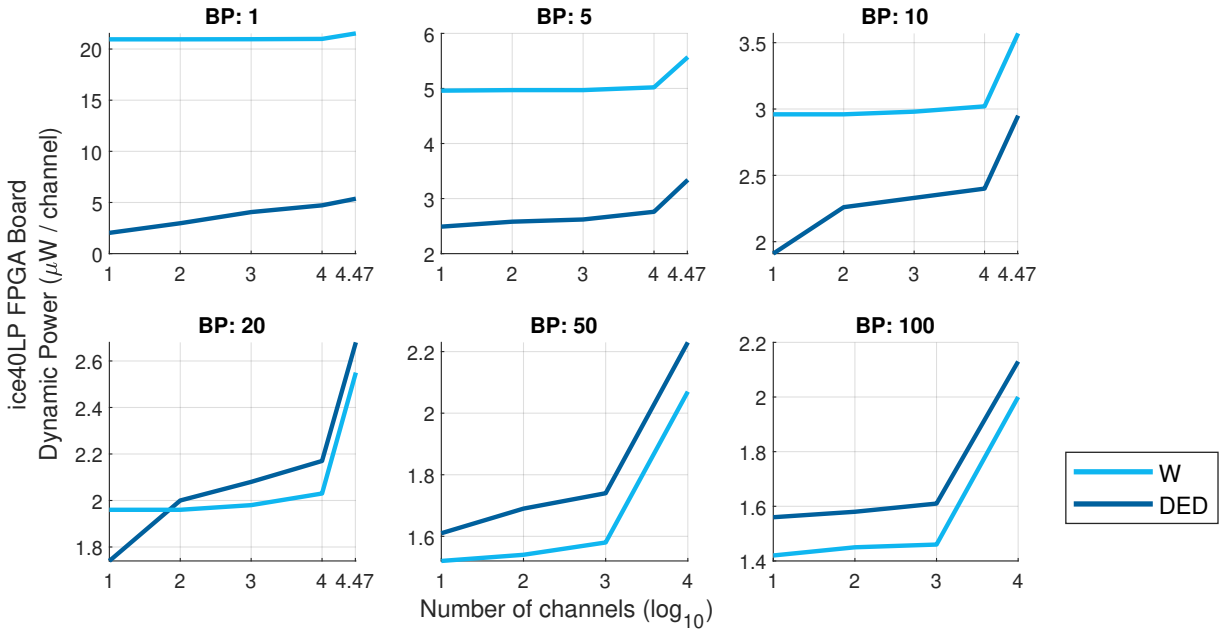


Figure 5.2: Total communication + processing power, i.e. dynamic power, for the windowed ('W') and DED compression schemes at a BP $\in$ [1, 5, 10, 20, 50, 100] ms. For each BP, S was respectively fixed as [2, 2, 2, 2, 3, 5]. For BPs/BTRs of 50 and 100 ms, the power is not given for 10000 and 30000 channels since the memory requirements exceeded the FPGA target memory budget.

5.3.1 Optimal Encoding Selection

From Fig. 5.2 and our analysis of the hardware costs in the Appendices Section C.4, for each BTR and n we selected the optimal compression system. These are given in Table 5.1 (a). We then determined each selected system's performance on the test data, i.e. data that had hereto been untouched. The resulting BRs are given in (b). The associated required number of FPGA logic cells and memory are given respectively in Table 5.1 (c) and (d). The total dynamic power on the Lattice ice40LP FPGA target, determined on the testing data, is given in (e). In summary, the DED encoding is suitable for short BTRs (less than approx. 20 ms) while the windowed encoding is preferred when the channel count or BTR is increased.

5.3.2 Testing the Encoding/Decoding

While having the BRs for each encoding and implementation is important, it is necessary to verify that each encoding functions as desired and is feasible. As such, encoding functions that encoded the binned and S-saturated MUA data according to each encoding and implementation, as well as with histogram-integrated versions, were implemented. The BRs were determined from the encoded data directly and were verified to match the analytically derived BRs. The full decodability of the data was then verified by decoding the original binned and S-saturated MUA data from the encodings and verifying that the original and decoded versions were a perfect match. This was done successfully for each method. The MATLAB code for each method and implementation's encoding and decoding is given at [107].

5.4 Discussion

5.4.1 Impact of Compression on Channel Counts in FPGA Target

From Table 5.1, knowing the compressed BRs per BP and channel count, we can derive how many channels can be hosted on-implant, for different FPGA board dimensions and BPs. We assume:

- A $10\,\mathrm{mW/cm^2}$ heat flux limit;
- An FPGA static power of $162\,\mu\mathrm{W}$;
- A separate power and hardware budget for the front-end amplifiers, filters and ADCs;
- A $20\,\mathrm{nJ/bit}$ communication energy;
- A binner processing power of $0.96\,\mu\mathrm{W}$, independent of BP.

Using the information from Table 5.1 (e), we can derive how many channels can be hosted on-implant while staying within implant power limits. This is compared to the number of channels that can be hosted given the standard uncompressed MUA representation at 1 ms BP, $S = 2$, that has a 1000 bps/channel BR.

The results are shown graphically in Fig. 5.3. It can be observed that between 4.6 and 26 times more MUA channels can be fit on-implant with data compression, depending on BP $\in \{1, 5, 10, 20, 50, 100\}$ ms and FPGA size $\in \{1, 2.5, 5, 7.5\}$ mm. As such, one can observe that data compression allows one to fit many more MUA channels onto the same implant. Therefore, the compression schemes developed in the last two chapters are a useful addition to MUA-based WI-BMIs.

5.4.2 Effects of Different Firing Rates and Communication Energy

As in the previous chapter, it is worth noting that the BRs are taken from 3 datasets that all target the motor cortex during hand kinematic tasks, using Utah arrays. It may be that different tasks, cortical regions and microelectrode arrays will have varying FRs, which will affect the measured BRs. Similarly, a $20\,\mathrm{nJ/bit}$ communication energy estimate is used, but this could be overly optimistic. Such values have been observed outside the neural medium, and communication energies within the neural medium are likely to be much higher. As such, while the processing power and resource estimates in this work are robust, the communication energy, made up of the BR and comm. energy per bit, may vary.

Cortical areas, microelectrode arrays and tasks that have higher MUA FRs will be better served, all else held equal, by the windowed scheme. This is because the windowed scheme works better if the FRs are non-sparse. However, higher communication energies will make the communication power dominate, and so the importance of the processing power, which is higher in the DED encoding, will be less important when choosing a system. It may be that, even with higher FRs than shown in this work, the DED scheme will be the best option given that it generally has lower BRs than the windowed scheme. This is especially true at finer BTRs.

Additionally, at higher communication energies, the role of data compression as a whole becomes more important. As such, the ratio of compressed to uncompressed channels that can be fit on-implant within heat limits, shown in Fig. 5.3, may be a conservative estimate. With significantly higher communication energies, a much larger ratio of compressed channels to uncompressed channels could be fit on-implant.

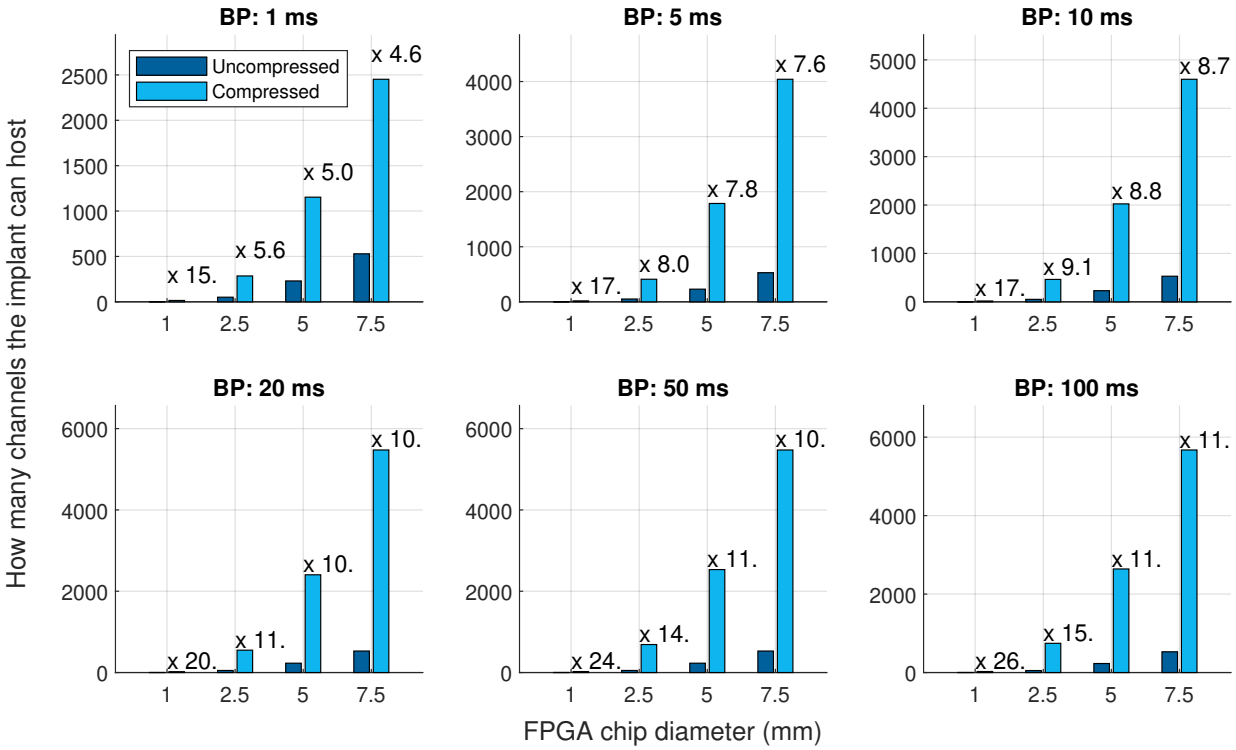


Figure 5.3: Number of MUA channels we can host on a Lattice ice40LP FPGA of different dimensions for the selected communication schemes at a BP $\in$ [1, 5, 10, 20, 50, 100] ms while remaining within the power budget. This is compared to the number of channels that can be hosted given the standard uncompressed MUA representation at 1 ms BP, $S = 2$, that has a 1000 bps/channel BR. The required power for any front-end ADC and pre-amplifiers is ignored. The total dynamic power budget is given by the FPGA dimensions multiplied by a heat flux limit of 10 mW/cm^2, assuming heat flux from both faces, minus the FPGA static power. For each BP and FPGA dimension pair, the ratio of how many more channels can be fitted on-implant with compression is also given.

5.4.3 Adaptive Huffman vs. Static Huffman encoding

While it is not shown in the main manuscript to simplify it, this work did not just look at SH encoding. For each of the four compression architectures, we also investigated modes with no lossless compression, with Adaptive Huffman encoding, which trains the Huffman encoders using the data collected on-implant in real time, and the entropic bandwidth, which gives the minimum BRs that could be achieved for each architecture given a perfect compression scheme. These modes are described in detail in the Appendix C, and the BR results given in Appendix C.2.

An interesting observation was that the SH encodings performed virtually identically to the Adaptive Huffman encodings, which were fully adaptive and assumed perfect knowledge of the to-be-compressed data. As such, we can conclude that SH encoders are a remarkably hardware-efficient and well-performing compression scheme for MUA data. This is because the shape of the to-be-compressed data is more or less known *a priori*, and so on-implant training is unnecessary.

5.4.4 Sample Histograms for Mapping

The use of sample histograms to add adaptivity into the SH encodings, as in the previous chapter, was also investigated. Also to simplify the manuscript, detailed discussion and the BR results are given in the Appendix C. It was found that sample histograms did improve the BRs, but that their hardware and processing power costs, while not very large, were noticeable. It was deemed that, for a 20 nJ/bit communication energy, the use of sample histograms was not justified for the reduction in communication power. However, with larger communication energies, the use of sample histograms may be warranted to reduce the BR and so the communication power, which may be the dominating factor.

5.5 Conclusion

In this chapter, we looked at various algorithms for compressing MUA data for WI-BMIs. We made significant use of SH encoders because of their hardware efficiency and good compression performance. We found that they performed exceptionally well, and that the MUA signal could be compressed to varying degrees depending on BP. For example, at the standard BP of 1 ms, between 4.6 and 26 times more channels can be fitted onto the same mm-scale implant merely because of the addition of the DED compression scheme proposed in this work.

It was found that for BPs $\leq$ 10 ms, the DED method had the lowest total processing + communication power and reduced the BR by up to almost an order of magnitude relative to the classical windowed method (e.g. to approx. 151 bps/channel for a BP of 1 ms and 1000 channels on-implant.). However, at larger BPs the windowed method performed best (e.g. approx. 29 bps/channel for a BP of 50 ms, independent of channel count).

As such, the work in this chapter can guide the choice of MUA data compression scheme for BMI applications, where the BR can be significantly reduced in hardware efficient ways. This enables the next generation of wireless intracortical BMIs, with small implant sizes, high channel counts, low-power, small hardware footprint, and good BTRs. MUA is already the most popular neural signal for WI-BMIs [14, 74, 77, 85]. The results from this work suggest that the MUA signal can be made even more attractive, given the possibility for hardware-efficient, ultra-low power compression. All code and results in this chapter have been made publicly available at [107].

Author Contributions

The work in this chapter was done in collaboration with Z.Z. Compression work in this chapter was carried out by O.W.S. Hardware design and processing power estimation work, given in Appendix C.3 was done by Z.Z.

Part II

Compressing Entire Spiking Activity

Chapter 6

Minimum Requirements for the Processing and Compression of ESA

This chapter addresses the hardware-efficient and low-bandwidth extraction of the ESA signal. While most modern WI-BMIs use the MUA signal, the ESA signal has been shown to have superior decoding ability. It is also, in theory, simple to extract on-implant, however this has not been investigated. Therefore in this chapter we looked at hardware optimisations for the on-implant extraction of the ESA signal. We found that in terms of its BDP and BR results, it often rivals the MUA signal while having a simpler extraction process.

6.1 Background

6.1.1 Entire Spiking Activity

Methods for the hardware-efficient and low power extraction of the MUA signal have been developed [68]. Furthermore, the compression of the MUA signal has been extensively explored in Part I of this thesis. However, the extraction and compression process of the MUA signal continues to be slightly complicated, involving highpassing the broadband, setting an appropriate, potentially adaptive spike-detection threshold [68], binning, and compression using advanced algorithms whose complexity depends on the required temporal resolution of the data (Chapter 5). As such, finding other, simple to extract intracortical signals with high BDP and low bandwidth would be beneficial.

The ESA signal is underexplored as a signal. However, recent work has shown that the ESA signal may contain the most behavioral information from amongst all common intracortical features [16]. By comparing the BDP of SUA, MUA, LFP and ESA signals, across a wide range of decoding algorithms, it was found that the ESA signal had the highest BDP [16]. It is generally extracted via highpass filtering of the broadband signal at approx. 300 Hz, followed by rectification and enveloping, and finally downsampling/binning. This process is shown in Fig. 6.1 in the context of WI-BMIs. Given the simple nature of the ESA extraction process and the high BDP it has been shown to have, it would be ideal if this signal could be extracted in a hardware-efficient way, with small bandwidth while maintaining high BDP. As such, in this chapter we investigate hardware-efficient and low-power means to extract and compress the ESA signal, in the hopes of obtaining an easy-to-extract, low BR, and high BDP signal for use in WI-BMIs.

Given our success with SH encoders for compressing the ESA signal in Chapter 3 and for compressing the

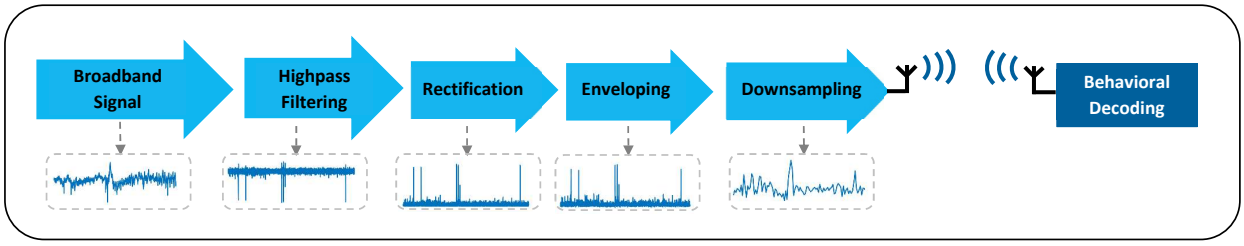


Figure 6.1: Typical ESA processing flow, shown for WI-BMIs.

MUA signal in Chapters 4 and 5, this chapter will also make use of SH encoders. It warrants mentioning that the ESA signal is significantly less sparse than the MUA signal. This is because values of 0 are significantly rarer due to the more analogue nature of the ESA signal, where background spiking activity is included. This reduced sparsity is obvious in comparing Fig. 4.1 and 6.2. As such, the windowed SH method was the main one used.

6.2 Methods

6.2.1 Analysed Data

For this chapter, we used the publicly available dataset from [65] (Sabes Raw Broadband from Table 2.4). It consists of raw neural intracortical broadband signals taken from a NHP subject with Utah arrays inserted in M1, with stored behavioral data. A random sample of 15 recordings were used as training data, and the remaining 12 as test data. The split is detailed in Table D.1 in Appendix D. Each recording has $n = 96$ channels.

6.2.2 ESA extraction process

6.2.2.1 Analog-to-Digital converter

The first processing step was to imitate an ADC. For both reduced power and hardware footprint, it would be convenient to have a low sampling rate and sample resolution. As such, we downsampled the resolution from the original 16 bits to b bits, and the frequency from 24.4 kHz to f kHz. The frequency downsampling was done via linear interpolation, and no anti-aliasing filter was used prior to downsampling. While this introduces distortion into the signal, it was found to be minor enough to have no effect on BDP. Furthermore, the lack of a lowpass filter greatly improved the hardware efficiency.

6.2.2.2 Highpass filtering and Rectification

When processing neural signals, it is common to use a classical highpass filter, e.g. Butterworth, Elliptic, Chebyshev, etc. However, recent work has found that a basic, extremely hardware efficient highpass filter can be made by simply delta-sampling the data [68]. As in econometrics, this removes long term trends from the data, i.e. low-frequency components. It is highly distorting in the frequency domain, however if the BDP does not suffer, the distortion is ultimately irrelevant for BMI purposes. It is as hardware efficient as possible for a highpass filter, consisting of a single subtraction operation.

Therefore, given our goal of achieving ultimate hardware efficiency, we opted for a simple delta-sampling operation as our highpass filter. Furthermore, the next step in processing flow is rectification, which is simply

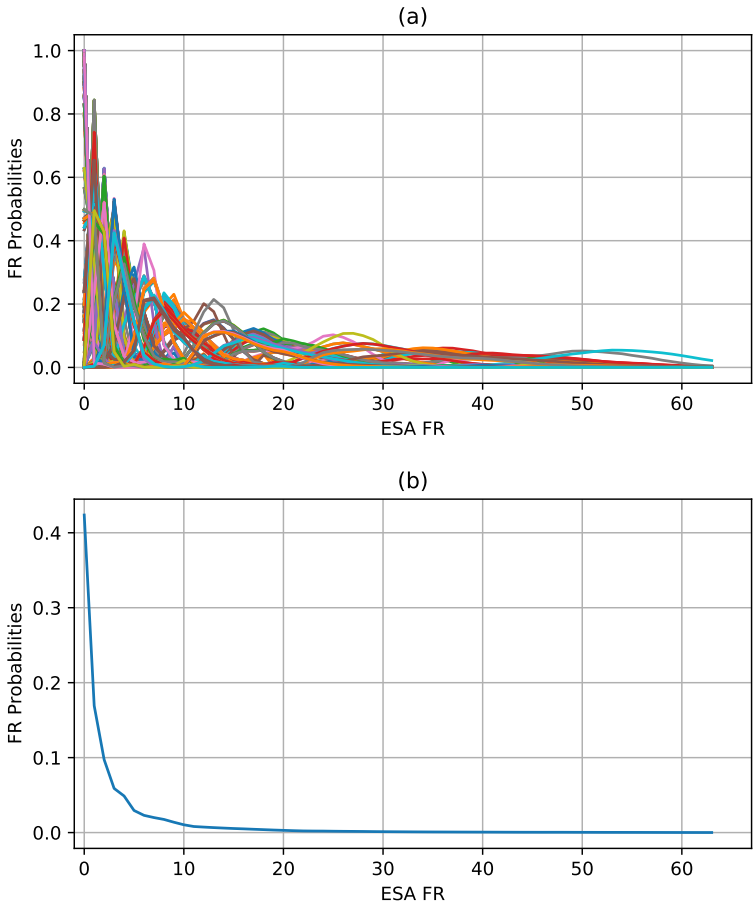


Figure 6.2: (a) A random sample of 1200 ESA grid search results' FR probability distributions, with BTR $\approx$ 100 ms. (b) Average ESA FR probability distribution across the above grid search results, with BTR $\approx$ 100 ms.

achieved by removing the signed bit, as such it does not require any operation to perform. The complete delta-sampling and rectification is given in Eq. 6.1.

$$y[k] = \begin{cases} x[k], & \text{if } k \leq h \\ \text{abs}(x[k] - x[k-h]), & \text{if } k > h \end{cases} \tag{6.1}$$

where y is the delta-sampled and rectified signal, x is the input signal, $k \in [\mathbb{Z}^+, 1 \leq k \leq p]$, and p is the length of the recording on one channel in samples. abs is the absolute transform, and $h \in \mathbb{Z}^+$ is an interleaving value. To reduce the dynamic range, only the l LSB of y were kept. In other words, we saturated i.e. thresholded the dynamic range of the rectified signal at l bits, so that $y \in [\mathbb{Z}^+, 0 \leq y \leq (2^l - 1)]$.

6.2.2.3 Enveloping and Downsampling

The next step in ESA processing is to envelope the rectified signal. Given that reducing the BR is a parallel goal, it makes sense to simultaneously downsample the frequency of the signal. The simplest method to accomplish both is to bin the signal with a non-overlapping window. By binning w samples together, this reduces the sampling rate of the data from f to f/w. This gives a BP, i.e. BTR, of w/f s, which is a useful metric for comparing the data flow to MUA data flows. The binned signal z is given by:

$$z[k_2] = \sum_{i=1}^{w} y[(k_2 - 1) \times w + i] \tag{6.2}$$

where $k_2 \in [\mathbb{Z}^+, 1 \leq k_2 \leq p/w]$.

To reduce the BR, it is useful to reduce the dynamic range of the signal. We did so by downsampling the resolution of the binned MUA signal, which consisted of keeping the m most significant bits of z, as this allowed us to keep the general shape of the data whilst doing away with the smaller details in the signal. Thresholding, as for the MUA signal in Part I, was also investigated for the ESA signal as a means of reducing the dynamic range. However, it was found to not perform well do to it overly negatively affecting the BDP.

6.2.2.4 Static Huffman compression with mapping

The next step was to integrate some lossless compression into the data flow. As throughout this thesis, SH encoders were used due to their hardware efficiency and good performance. As in Chapter 5, only 1 encoder was implemented on-implant, due to the result in Chapter 4 for the MUA signal that having 1 encoder often gave optimal compression in a very hardware efficient way. Similarly, here the SH encoder was trained on a decaying exponential so that smaller values received shorter codewords, with an Sorted Codeword Length Vector (SCLV) of [1, 2, 3, ..., $S-1$, $S-1$].

SH encoders work best if the data is sorted so that the most common symbols index the shortest codewords, as in Table 2.2 (b). As such, some way to estimate the on-implant ESA histogram is needed to make full use of pre-trained SH encoders. In previous chapters, we used a sample histogram to improve the adaptivity of the SH encoders for MUA data (Fig. 4.3). In this chapter, we adopted the same strategy for the ESA signal.

6.2.2.4.1 Losslessly compressing the data

The first $2^s - 1$ samples from each channel were used to train the histogram, where s was the size of each histogram bin in bits. Once the given number of samples had been used to populate the histogram, the mapping was produced using the same pseudo-sorting algorithm from Chapter 4, detailed in Fig B.10 in the

Appendices. The rest of the data was then mapped to its new values. The hope is that the more common symbols will be mapped to smaller values. Those values are then compressed using the SH encoder defined above, which gives shorter codewords to smaller values. Again, this process is shown in Fig. 4.3. If $s = 0$, then no histogram or mapping was used and the data was treated as if it was already sorted.

6.2.2.4.2 Calculating the BR

For each parameter combination, using the processed data, the final compressed BR is analytically given by:

$$\mathrm{BR} = \frac{\sum_{i=0}^{2^m-1}\left(\mathrm{SCLV}_{i+1} \times \sum_{c=1}^{n}\sum_{k=1}^{p/w}(Z_{c,k} == i)\right)}{n \times f/w} \qquad \text{[bps/channel]} \quad (6.3)$$

where $(Z_{c,k} == i)$ has a boolean value of 1 if the k^{th} element of the processed ESA data on channel $c \in [\mathbb{Z}, 1 \leq c \leq n = 96]$ is equal to i, and a value of 0 otherwise. Additionally, $\mathrm{SCLV_i}$ is the length of the i^{th} codeword, i.e. the i^{th} element of the SCLV of the aforementioned SH encoder.

6.2.2.5 Behavioral Decoding

The final step is to ensure that the processing has not eliminated too much useful data. This is done by using the processed ESA signal (without the lossless compression from the previous section) to decode the tracked behavior. To do so, we used a Wiener Cascade Filter decoder to decode the X and Y-axis cursor velocities associated with this neural dataset. For each processing flow parameter combination, the decoder was hyper-parameter optimised. These hyper-parameters included the Wiener Cascade Filter α value, the degree of the Wiener Cascade Filter, a lag parameter that specified the lag between the kinematic and neural data, the amount of timesteps (i.e. taps) the Wiener Cascade Filter considered, and the width of a causal moving average filter used to smooth the neural data prior to being feed into the decoder. l_2 (also known as ridge or Tikhonov) regularisation was used. The hyper-parameter optimisation was done via 5-fold cross-validation, with a 80-20% training-testing split. The final BDP was then measured on the testing data, using the hyper-parameters determined from the 5-fold cross-validation.

6.2.3 Parameter Optimisation

The objective was to find a combination of b, f, h, l, w, m and s that maximised BDP while minimizing the required hardware and BR. As such, a grid search was used to iterate through different parameter combinations. An initial narrow set of parameters was scanned, and it was found that h had little effect, and so was set to the best overall performing value of $h = 3$.

Next, a broad set of parameters was iterated through each recording of the training data. These parameters are shown in Table 6.1. For each parameter combination, hyper-parameters were optimised using 5-fold cross-validation, and the final BDP on the testing set was taken as the BDP for that parameter combination. For each parameter combination, the BDP and BR were then stored, along with the final BTR. It may be worth mentioning that the histogram size s does not affect the BDP and only impacts the BR and hardware resources, since it affects the lossless compression.

6.3 Results

The grid search results were plotted in Fig. 6.3 as a 2D colored scatter plot, where BDP was shown as a function of BR and the BTR of the data for each parameter combination.

Table 6.1: ESA grid search parameter space.

ESA Extraction Parameters		**Description**
f **(kHz)**	4, 5, 6, 7, 8, 9	Sampling frequency
b **(bits)**	8, 9, 10, 11, 12	Sampling resolution
h	3	Interleaving value
l **(bits)**	3, 4, 5, 6	Dynamic range of rectified signal
w **(samples)**	2**[7, 8, 9, 10, 11, 12]	Binning window
m **(bits)**	4, 5, 6, 7	Nb. of most significant bits to keep in the final signal
s **(bits/bin)**	0, 1, 2, 3, 4, 5	Histogram size

Behavioral Decoder Hyper-parameters		**Description**
α	0, 10^{-2}, 10^{-3}	Wiener Cascade Filter hyper-parameter
Degree	2, 3, 5	Wiener Cascade Filter hyper-parameter
Timesteps	5, 10, 15	How many samples (i.e. taps) to use.
Lag	0, 10 times f/w	Introduced lag between neural and kinematic data.
Moving Average Window (ms)	0, 50, 100	Causal window width

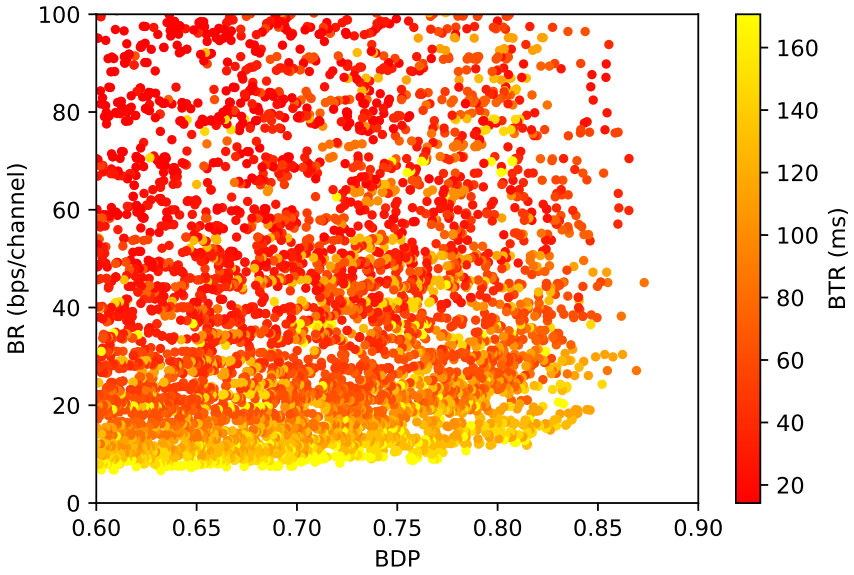


Figure 6.3: Grid search results from Table 6.1. Each parameter combination is represented by a data point, and is plotted as a function of its BR, BDP and BTR.

The results are encouraging, in that BDPs above 80% were achieved at BRs below 20 bps/channel. However, generally speaking, a lower BR meant a higher BTR. One would prefer a low BTR, where we have a high temporal resolution for the data, and also a low BR. As such, there is an unfortunate but intuitive tradeoff. Sending out the data at a low temporal resolution reduces the bandwidth. Similarly, a higher temporal resolution requires a higher BR.

6.3.1 Comparing the training data ESA results to state-of-the-art MUA results

Chapters 4 and 5 included the same Sabes dataset, for the compression of the MUA signal, as in this chapter for the processing of ESA. As such, we can make a direct comparison of the exact same recordings, comparing the MUA and ESA BRs, BTRs and BDPs. This is important, as there can be significant variation in the BDP and BR between recordings, and so making comparisons between the same recordings removes confounding factors. Therefore, for each occurring BTR value, we selected a parameter combination for the ESA signal that had a good trade-off between BDP and BR. We then looked at the MUA results from Table 5.1, and chose MUA processing parameters and encodings that were found to be optimal across the 3 datasets, for BTRs $\in [1, 5, 10, 20, 50, 100]$ ms.

In Chapter 5, the MUA compression results depend on the number of channels, where having more channels sometimes increases the BR. However, there were only 15 training recordings used in this work, with 96 channels each. As such, a maximum of 1440 channels were available. As such, we considered having 1000 channels on-implant, and used the selection from Table. 5.1 in Chapter 5 for 1000 channels. As such, for BTRs $\in [1, 5, 10, 20, 50, 100]$ ms, we selected encoding schemes [DED, DED, DED, W, W, W] with MUA $S = [2, 2, 2, 2, 3, 5]$ respectively.

As such, Fig. 6.4 shows the ESA vs. MUA BDPs as a function of BTR and BR. Each data point is a parameter combination, and the BRs and BDPs are averaged across all training recordings, for both the MUA and ESA results. We can observe a similar trend for both signals in decreasing BR being associated with increased BTR. However, the MUA signal seems to have a lower BR for a similar BTR, for BTRs $\leq$ approx 50 ms, with a slightly higher BDP. Contrary to that, at BTRs $>$ approx. 50 ms, the ESA signal has virtually identical BR for the same BTR as the MUA signal. The ESA signal also seems to have a slightly better BDP for the same BR and BTR, making it perform slightly better at BTRs $\geq$ approx. 50 ms.

Given the findings in Table 5.1, we can also determine that the MUA signal would perform even better at fewer than 1000 channels, but worse at more than 1000 channels at BPs equal or less than 20 ms.

6.3.2 Test data results

Finally, we tested whether the system performed as expected on new data, as all results may have been overfitted to the training parameter selection. To do so, we took the same ESA extraction parameters from Fig. 6.4 (given in Supplemental Material, Table 2), and tested their performance on the testing data.

The ESA testing data BDPs, BTRs and BRs were calculated for the parameters and these are plotted vs. the MUA test data. The MUA test data was compressed using the same parameters and algorithms as the MUA train data, detailed in Table 5.1 of Chapter 5, and the MUA BDP averaged across the test recordings from the MUA BDP results from Chapter 5.

The test recording results are shown in Fig. 6.5. We can observe that the BDPs in the test data are somewhat lower than in the training data. However, the same relationship between ESA and MUA data holds in the new test data as it does in the train data. This is encouraging, as it shows that the ESA processing system is performing consistently and well on the new data. This shows that the results in Fig. 6.4 are not merely the result of overfit to a broad parameter search.

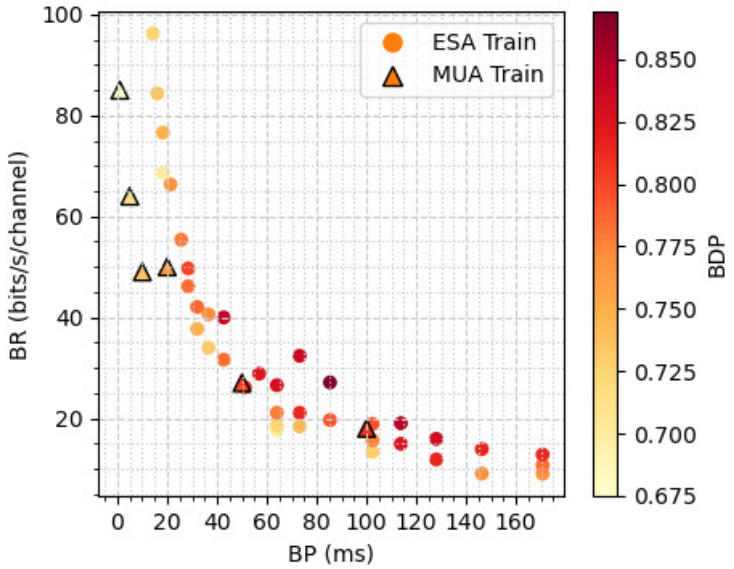


Figure 6.4: BDP as a function of BTR and BR, for both ESA results in this work and the MUA results using the same recordings. All results are from the training data. Triangle markers with black outline represent MUA results, and circular markers without outlines represent ESA results. We can observe a similar trend for both signals in decreasing BR being associated with increased BTR. The MUA signal seems to perform slightly better than the ESA signal at BTRs $<$ approx. 50 ms, and the ESA signal seems to perform better at BTRs $\geq$ approx. 50 ms. The processing parameters for the ESA signal for each data point are given in Appendix, Table D.1.

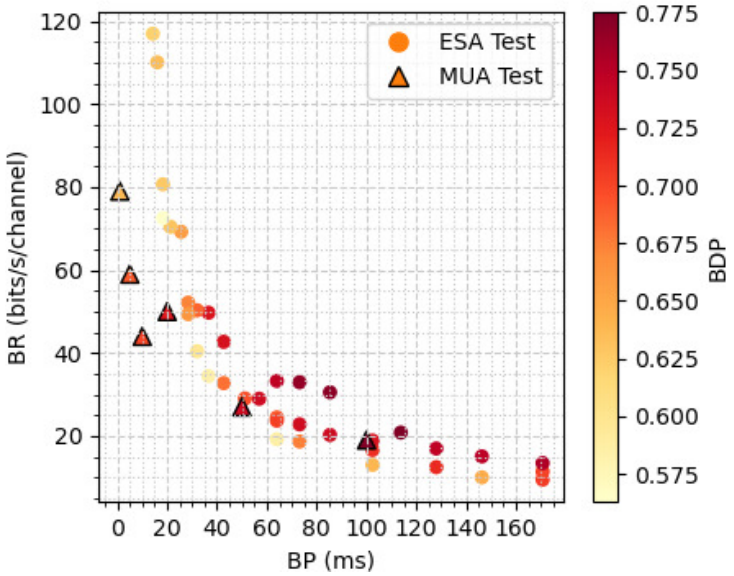


Figure 6.5: BDP as a function of BTR and BR, for both ESA test results in this work and the MUA results (from the recordings referred to in this work as test recordings) from Chapter 5. Triangle markers with black outline represent MUA test results, and circular markers without outlines represent ESA test results. We can observe a similar trend for both signals as in Fig.6.4.

6.4 Discussion

6.4.1 Whether to select MUA or ESA depends on the desired BTR

The MUA signal processing flow is much more complex than the ESA processing flow. This is because a highly-optimised MUA processing flow involves high-pass filtering, setting an appropriate and potentially adaptive threshold for spike-detection, and then the binning and compression with a simple synchronous or complex delta-asynchronous algorithm depending on BTR and channel count [59,68]. The ESA flow involves only high-pass filtering and rectification via a single subtraction operation, binning via a summing operation, and a simple compression algorithm. The compression scheme for the ESA signal in this chapter is in fact the same windowed scheme as used for the MUA data in Chapter 5 for BTRs $\geq$ 20 ms, with the addition of a sample histogram. As such, considering that the ESA processing flow is simpler than the MUA processing flow, the ESA signal seems superior to the MUA signal for WI-BMIs at BTRs $\geq$ approx. 50 ms. This is because, at BTRs $\geq$ approx. 50 ms, the ESA gives similar BR and BDP results as the MUA signal for the same BTR, with a simpler data flow.

For BTRs below 50 ms, the MUA signal appears to be more appropriate, depending on whether the additional complexity of the MUA signal processing scheme [68] is acceptable or not.

6.4.2 Each channel sharing the same histogram/mapping

It was also considered that each ESA channel could share the same histogram and mapping. This would save on RAM memory. However, the number of logic circuits would be the same. This is because, even if every channel has its own histogram, the histogram module's use is multiplexed and so only one histogram module exists. It was found that the compression performance was significantly better if each channel had its own histogram. As such, a design choice was made to have every channel have its own histogram, even if this requires more RAM.

6.4.3 Further optimisation

There are a number of other operations we could consider to improve the ESA processing flow. These include:

- We could investigate other SH encoder distributions (other than a decaying exponential).
- Rather than selecting the most significant bits to downsample the data, we could look at applying a Discrete Wavelet Transform, and compressing the coefficients.
- One could have multiple SH encoders. Each channel's histogram could be used for assignment of channels to their optimal encoders, as in [59]. This increases hardware complexity, but may improve BR somewhat.
- One could map the final pre-compression ESA values to a custom non-linear scale to reduce the dynamic range and BR, while maximising the BDP as a function of BR. For example, one could use each channel's histogram to see what values occur on each channel. One could then communicate out only those values as possibilities, reducing the dynamic range by customising it to each channel's signal.
- In Chapter 3, it was shown that LNNT pre-processing was an effective step for improving the compression performance of SH encoders. In particular, with enough training data, it may perform effectively. It is relatively simple to implement in hardware, especially if the weights are powers of 2 and so the

multiplication can be replaced with bit-shifting. However, there was not enough time to explore further pre-processing methods. As such, while this thesis presents evidence that LNNT pre-processing can improve ESA compression, it has not been analysed at the same level that the rest of the processing methods in this chapter have been.

- In Chapter 3, we determined that clipping protects against overfit in the case of large dynamic ranges. If the dynamic range of thew ESA signal cannot be reduced sufficiently, clipping may be an interesting method to make the compression more reliable.

6.4.4 Hardware complexity suffers from large dynamic range

The hardware cost of the ESA compression is made worse by large dynamic ranges. For example, the pseudo-sort algorithm for mapping the ESA values to codewords requires an LUT of dimensions $S \times S$. As such, the dynamic range S of the ESA signal should be minimised as much as possible. Its dynamic range is currently unattractive, being significantly larger than that of the MUA signal, where $S = 2^m$ given in Table 6.1.

6.5 Conclusion

In this chapter, we looked at optimising the intracortical ESA signal for behavioral decoding BMI applications. We found that the signal can be highly optimised, to the point it rivals the bandwidth and decoding performance of the most common intracortical signal in WI-BMIs, MUA. Additionally, the ESA on-implant processing flow is significantly simpler. Further optimisations of the ESA processing flow are also likely possible. As such, the ESA signal appears to be a strong candidate target for modern WI-BMI applications. It gives high behavioral decoding performance at decent temporal resolutions and low bandwidths. It also has an extremely hardware efficient on-implant extraction procedure. The dynamic range and its effect on BRs is a continued issue that warrants further research. However, optimism is absolutely warranted: this chapter shows that merely delta-sampling the broadband signal, and binning the result, gives state-of-the-art behavioral decoding performance at reasonable BRs and BTRs.

Chapter 7

Conclusion and Future Directions

As BMIs develop, a remaining bottleneck for widespread clinical application is wireless powering and communication. Percutaneous connections are simply not a tangible solution for chronic BMI use. However, wireless communication consumes significant power. This is especially true considering that every μW of power used on-implant first has to be wirelessly transferred to it. This transfer is highly lossy, which only further heats the tissue. As such, heat flux safety limits into cortical tissue are an immediate constraint for WI-BMIs. Therefore, techniques to extract, process and compress intracortical neural signals in ultra-low power ways are needed. Furthermore, given the extremely hardware constrained WI-BMI environment, any such processing must be performed using minimal computation.

In this thesis, we tackled this problem. We showed that intracortical data compression is indeed possible with minimal additional computation, even within the extreme hardware constraints.

7.1 Original contributions

This thesis contains multiple original contributions.

7.1.1 MUA compression

In Part I, the popular MUA signal was compressed by approximately an order of magnitude relative to the general standard in hardware-efficient ways. Multiple compression schemes were introduced, such as combining SH encoders with the common windowing method, as well as advanced methods such as the Delta Event-Driven and Group Event-Driven encodings. Each of these methods was analysed intensively, and a collaboration was undertaken with Z.Z. to produce the hardware designs of each. Furthermore, various additional techniques were introduced to improve the performance of SH encoders, such as sample histograms to map values along with a pseudo-sort algorithm, the use of multiple encoders with assignment, and clipping to protect against overfit in the case of large dynamic ranges.

A final recommendation of the best MUA compression schemes for different scenarios was given in Table 5.1, based on results from multiple NHP subjects. All together, this thesis contributes extremely hardware-efficient means of compressing MUA signals with good BTRs and BDP. For example, one could keep a 1 ms BTR and obtain a lossless 151 bps/channel BR, relative to the standard 1 kbps/channel for 1000 recording channels. At a 20 ms BTR, one could reduce the BR down to 50 bps/channel without any degradation in behavioral decoding quality.

7.1.2 ESA compression

The ESA signal is underexplored in BMI work, but recent work demonstrates its state-of-the-art decoding ability [16]. This thesis introduces extremely hardware-efficient means of extracting and compressing the signal. The basic flow involves delta-sampling as a rudimentary highpass filter, combined with binning for simultaneous enveloping and downsampling. Finally, the windowed compression scheme from the MUA work was applied to the ESA signal, with good results. Various pre-processing compression techniques were also explored, such as LNNT encoding. Overall, it is remarkable that simply delta-sampling the broadband signal, and binning the result, can reliably produce a signal with state-of-the-art behavioral decoding performance. Further investigation is absolutely warranted.

7.2 Future directions

It is the opinion of the author that future work should target two areas:

1. The ESA signal. It has remarkable decoding ability and robustness, and it presents interesting research opportunities. Improvements to its hardware-efficient extraction, beyond what was accomplished in this thesis, are an attractive prospect. Furthermore, developing custom compression schemes, e.g. applying LNNT along with sample histograms, could bring significant benefit. Overall, the ESA signal has significant potential and continues to be underexplored as a signal in BMI research. I believe that, in time, it is likely it will uproot MUA as the standard target signal in iBMIs thanks to its simple extraction process, robust chronic performance and state-of-the-art BDP.

2. The exploration of implementing MUA compression schemes in ASIC hardware. Of the work on different signals targeted in this thesis, the work on MUA compression is the most mature. However, there is significant space to explore translation of the MUA encoding methods to ASIC hardware. ASICs can tailor their hardware designs to the required computations, allowing significant improvements in power and space compared to FPGAs. Furthermore, in ASICs the use of memory can be customised, and the front-end processing of the MUA signal combined with the compression module in innovative, yet-to-be-explored ways. Overall, this would facilitate the practical implementation of MUA extraction and compression schemes into iBMIs as a whole, helping solve the obstacle to widespread clinical use of iBMIs.

7.3 Concluding remarks

I am without doubt that iBMIs will be wireless, and that their clinical adaptation depends on it. The degree to which they can scale will depend on how they can limit the communication power. Therefore, hardware and power efficient means to process and compress the signal on-implant, prior to communication, are a must. In line that that goal, this thesis showed that such processing and compression was indeed possible within hardware and power limits. The reductions in communication power were significant, and enabled large increases in the number of recording channels while remaining within safety limits of intracortical heat flux. Overall, safer, more powerful BMIs result from smaller bandwidth, and I believe this thesis made significant strides in that direction.

Bibliography

[1] Christopher H Merritt, Matthew A Taylor, Caleb J Yelton, and Swapan K Ray. Economic impact of traumatic spinal cord injuries in the united states. *Neuroimmunology and neuroinflammation*, 6, 2019.

[2] Kim D Anderson. Targeting recovery: priorities of the spinal cord-injured population. *Journal of neurotrauma*, 21(10):1371–1383, 2004.

[3] Christine H Blabe, Vikash Gilja, Cindy A Chestek, Krishna V Shenoy, Kim D Anderson, and Jaimie M Henderson. Assessment of brain–machine interfaces from the perspective of people with paralysis. *Journal of neural engineering*, 12(4):043002, 2015.

[4] Jacob Lahr, Christina Schwartz, Bernhard Heimbach, Ad Aertsen, Jörn Rickert, and Tonio Ball. Invasive brain–machine interfaces: a survey of paralyzed patients' attitudes, knowledge and methods of information retrieval. *Journal of Neural Engineering*, 12(4):043001, 2015.

[5] Johan Wessberg, Christopher R Stambaugh, Jerald D Kralik, Pamela D Beck, Mark Laubach, John K Chapin, Jung Kim, S James Biggs, Mandayam A Srinivasan, and Miguel AL Nicolelis. Real-time prediction of hand trajectory by ensembles of cortical neurons in primates. *Nature*, 408(6810):361–365, 2000.

[6] Mijail D Serruya, Nicholas G Hatsopoulos, Liam Paninski, Matthew R Fellows, and John P Donoghue. Instant neural control of a movement signal. *Nature*, 416(6877):141–142, 2002.

[7] Dawn M Taylor, Stephen I Helms Tillery, and Andrew B Schwartz. Direct cortical control of 3D neuroprosthetic devices. *Science*, 296(5574):1829–1832, 2002.

[8] JD Simeral, Sung-Phil Kim, MJ Black, JP Donoghue, and LR Hochberg. Neural control of cursor trajectory and click by a human with tetraplegia 1000 days after implant of an intracortical microelectrode array. *Journal of neural engineering*, 8(2):025027, 2011.

[9] Daniel Bacher, Beata Jarosiewicz, Nicolas Y Masse, Sergey D Stavisky, John D Simeral, Katherine Newell, Erin M Oakley, Sydney S Cash, Gerhard Friehs, and Leigh R Hochberg. Neural point-and-click communication by a person with incomplete locked-in syndrome. *Neurorehabilitation and neural repair*, 29(5):462–471, 2015.

[10] Jose M Carmena, Mikhail A Lebedev, Roy E Crist, Joseph E O'Doherty, David M Santucci, Dragan F Dimitrov, Parag G Patil, Craig S Henriquez, Miguel A L Nicolelis, and Idan Segev. Learning to control a brain–machine interface for reaching and grasping by primates. *PLoS biology*, 1(2):e42, 2003.

[11] Leigh R Hochberg, Mijail D Serruya, Gerhard M Friehs, Jon A Mukand, Maryam Saleh, Abraham H Caplan, Almut Branner, David Chen, Richard D Penn, and John P Donoghue. Neuronal ensemble control of prosthetic devices by a human with tetraplegia. *Nature*, 442(7099):164–171, 2006.

[12] Jennifer L Collinger, Brian Wodlinger, John E Downey, Wei Wang, Elizabeth C Tyler-Kabara, Douglas J Weber, Angus JC McMorland, Meel Velliste, Michael L Boninger, and Andrew B Schwartz. High-performance neuroprosthetic control by an individual with tetraplegia. *The Lancet*, 381(9866):557–564, 2013.

[13] B Wodlinger, JE Downey, EC Tyler-Kabara, AB Schwartz, ML Boninger, and JL Collinger. Ten-dimensional anthropomorphic arm control in a human brain- machine interface: difficulties, solutions, and limitations. *Journal of neural engineering*, 12(1):016011, 2014.

[14] Adrien B Rapeaux and Timothy G Constandinou. Implantable brain machine interfaces: first-in-human studies, technology challenges and trends. *Current Opinion in Biotechnology*, 72:102–111, 2021. Tissue, Cell and Pathway Engineering.

[15] Claude Elwood Shannon. A mathematical theory of communication. *Bell Syst. Tech. J.*, 27(3):379–423, 1948.

[16] Nur Ahmadi, Timothy G Constandinou, and Christos-Savvas Bouganis. Robust and accurate decoding of hand kinematics from entire spiking activity using deep learning. *Journal of Neural Engineering*, 18(2):026011, 2021.

[17] Chethan Pandarinath, Paul Nuyujukian, Christine H Blabe, Brittany L Sorice, Jad Saab, Francis R Willett, Leigh R Hochberg, Krishna V Shenoy, and Jaimie M Henderson. High performance communication by people with paralysis using an intracortical brain-computer interface. *Elife*, 6:e18554, 2017.

[18] Francis R Willett, Donald T Avansino, Leigh R Hochberg, Jaimie M Henderson, and Krishna V Shenoy. High-performance brain-to-text communication via handwriting. *Nature*, 593(7858):249–254, 2021.

[19] Edward V Evarts. Relation of pyramidal tract activity to force exerted during voluntary movement. *Journal of neurophysiology*, 31(1):14–27, 1968.

[20] Jacques J Vidal. Real-time detection of brain events in EEG. *Proceedings of the IEEE*, 65(5):633–641, 1977.

[21] P R Kennedy and R AE Bakay. Restoration of neural output from a paralyzed patient by a direct brain connection. *Neuroreport*, 9(8):1707–1711, 1998.

[22] Niels Birbaumer, Nimr Ghanayim, Thilo Hinterberger, Iver Iversen, Boris Kotchoubey, Andrea Kübler, Juri Perelmouter, Edward Taub, and Herta Flor. A spelling device for the paralysed. *Nature*, 398(6725):297–298, 1999.

[23] John K Chapin, Karen A Moxon, Ronald S Markowitz, and Miguel AL Nicolelis. Real-time control of a robot arm using simultaneously recorded neurons in the motor cortex. *Nature neuroscience*, 2(7):664–670, 1999.

[24] Nicholas Hatsopoulos, Jignesh Joshi, and John G O'Leary. Decoding continuous and discrete motor behaviors using motor and premotor cortical ensembles. *Journal of neurophysiology*, 92(2):1165–1174, 2004.

[25] Nicholas G Hatsopoulos and John P Donoghue. The science of neural interface systems. *Annual review of neuroscience*, 32:249, 2009.

[26] Leigh R Hochberg, Daniel Bacher, Beata Jarosiewicz, Nicolas Y Masse, John D Simeral, Joern Vogel, Sami Haddadin, Jie Liu, Sydney S Cash, Patrick Van Der Smagt, et al. Reach and grasp by people with tetraplegia using a neurally controlled robotic arm. *Nature*, 485(7398):372–375, 2012.

[27] A Bolu Ajiboye, Francis R Willett, Daniel R Young, William D Memberg, Brian A Murphy, Jonathan P Miller, Benjamin L Walter, Jennifer A Sweet, Harry A Hoyen, Michael W Keith, et al. Restoration of reaching and grasping movements through brain-controlled muscle stimulation in a person with tetraplegia: a proof-of-concept demonstration. *The Lancet*, 389(10081):1821–1830, 2017.

[28] James J Jun, Nicholas A Steinmetz, Joshua H Siegle, Daniel J Denman, Marius Bauza, Brian Barbarits, Albert K Lee, Costas A Anastassiou, Alexandru Andrei, Çağatay Aydın, et al. Fully integrated silicon probes for high-density recording of neural activity. *Nature*, 551(7679):232–236, 2017.

[29] Nicholas A Steinmetz, Cagatay Aydin, Anna Lebedeva, Michael Okun, Marius Pachitariu, Marius Bauza, Maxime Beau, Jai Bhagat, Claudia Böhm, Martijn Broux, et al. Neuropixels 2.0: A miniaturized high-density probe for stable, long-term brain recordings. *Science*, 372(6539):eabf4588, 2021.

[30] Mikhail A Lebedev and Miguel AL Nicolelis. Brain-machine interfaces: From basic science to neuroprostheses and neurorehabilitation. *Physiological reviews*, 97(2):767–837, 2017.

[31] David M Brandman, Sydney S Cash, and Leigh R Hochberg. human intracortical recording and neural decoding for brain–computer interfaces. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 25(10):1687–1696, 2017.

[32] Chad E Bouton, Ammar Shaikhouni, Nicholas V Annetta, Marcia A Bockbrader, David A Friedenberg, Dylan M Nielson, Gaurav Sharma, Per B Sederberg, Bradley C Glenn, W Jerry Mysiw, et al. Restoring cortical control of functional movement in a human with quadriplegia. *Nature*, 533(7602):247–250, 2016.

[33] Elena Moro. Complications of DBS surgery—insights from large databases. *Nature Reviews Neurology*, 12(11):617–618, 2016.

[34] David K Piech, Benjamin C Johnson, Konlin Shen, M Meraj Ghanbari, Ka Yiu Li, Ryan M Neely, Joshua E Kay, Jose M Carmena, Michel M Maharbiz, and Rikky Muller. A wireless millimetre-scale implantable neural stimulator with ultrasonically powered bidirectional communication. *Nature Biomedical Engineering*, 4(2):207–222, 2020.

[35] Yan Wang, Santhisagar Vaddiraju, Bing Gu, Fotios Papadimitrakopoulos, and Diane J Burgess. Foreign body reaction to implantable biosensors: effects of tissue trauma and implant size. *Journal of diabetes science and technology*, 9(5):966–977, 2015.

[36] Jihun Lee, Ethan Mok, Jiannan Huang, Lingxiao Cui, Ah-Hyoung Lee, Vincent Leung, Patrick Mercier, Steven Shellhammer, Lawrence Larson, Peter Asbeck, et al. An implantable wireless network of distributed microscale sensors for neural applications. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 871–874. IEEE, 2019.

[37] Dongjin Seo, Jose M Carmena, Jan M Rabaey, Elad Alon, and Michel M Maharbiz. Neural dust: An ultrasonic, low power solution for chronic brain-machine interfaces. *arXiv preprint arXiv:1307.2196*, 2013.

[38] Nur Ahmadi, Matthew L Cavuto, Peilong Feng, Lieuwe B Leene, Michal Maslik, Federico Mazza, Oscar Savolainen, Katarzyna M Szostak, Christos-Savvas Bouganis, Jinendra Ekanayake, et al. Towards a distributed, chronically-implantable neural interface. In *2019 9th International IEEE/EMBS Conference on Neural Engineering (NER)*, pages 719–724. IEEE, 2019.

[39] Jihun Lee, Vincent Leung, Ah-Hyoung Lee, Jiannan Huang, Peter Asbeck, Patrick P Mercier, Stephen Shellhammer, Lawrence Larson, Farah Laiwalla, and Arto Nurmikko. Neural recording and stimulation using wireless networks of microimplants. *Nature Electronics*, 4(8):604–614, 2021.

[40] Xilin Liu, Milin Zhang, Basheer Subei, Andrew G Richardson, Timothy H Lucas, and Jan Van der Spiegel. The pennbmbi: Design of a general purpose wireless brain-machine-brain interface system. *IEEE transactions on biomedical circuits and systems*, 9(2):248–258, 2015.

[41] Ming Yin, David A Borton, Juan Aceros, William R Patterson, and Arto V Nurmikko. A 100-channel hermetically sealed implantable device for chronic wireless neurosensing applications. *IEEE transactions on biomedical circuits and systems*, 7(2):115–128, 2013.

[42] John D Simeral, Thomas Hosman, Jad Saab, Sharlene N Flesher, Marco Vilela, Brian Franco, Jessica Kelemen, David M Brandman, John G Ciancibello, Paymon G Rezaii, et al. Home use of a wireless intracortical brain-computer interface by individuals with tetraplegia. *medRxiv*, 2019.

[43] Andy Zhou, Samantha R Santacruz, Benjamin C Johnson, George Alexandrov, Ali Moin, Fred L Burghardt, Jan M Rabaey, Jose M Carmena, and Rikky Muller. A wireless and artefact-free 128-channel neuromodulation device for closed-loop stimulation and recording in non-human primates. *Nature biomedical engineering*, 3(1):15–26, 2019.

[44] Patrick D Wolf and WM Reichert. Thermal considerations for the design of an implanted cortical brain–machine interface (BMI). *Indwelling Neural Implants: Strategies for Contending with the In Vivo Environment*, pages 33–38, 2008.

[45] Joseph C LaManna, Kimberly A McCracken, Madhavi Patil, and Otto J Prohaska. Stimulus-activated changes in brain tissue temperature in the anesthetized rat. *Metabolic brain disease*, 4(4):225–237, 1989.

[46] IEEE C95. 1. IEEE standard for safety levels with respect to human exposure to electric, magnetic, and electromagnetic fields, 0 Hz to 300 GHz. The Institute of Electrical and Electronics Engineers New York, NY, 2019.

[47] Arto Nurmikko. Challenges for large-scale cortical interfaces. *Neuron*, 108(2):259–269, 2020.

[48] Keyoor Gosalia, James Weiland, Mark Humayun, and Gianluca Lazzi. Thermal elevation in the human eye and head due to the operation of a retinal prosthesis. *IEEE Transactions on Biomedical Engineering*, 51(8):1469–1477, 2004.

[49] Pavel S Yarmolenko, Eui Jung Moon, Chelsea Landon, Ashley Manzoor, Daryl W Hochman, Benjamin L Viglianti, and Mark W Dewhirst. Thresholds for thermal damage to normal tissues: an update. *International Journal of Hyperthermia*, 27(4):320–343, 2011.

[50] Sebastian Stoecklin, Adnan Yousaf, Gunnar Gidion, and Leonhard Reindl. Efficient wireless power transfer with capacitively segmented rf coils. *IEEE Access*, 8:24397–24415, 2020.

[51] Chul Kim, Jiwoong Park, Sohmyung Ha, Abraham Akinin, Rajkumar Kubendran, Patrick P. Mercier, and Gert Cauwenberghs. A 3 mm $\times$ 3 mm fully integrated wireless power receiver and neural interface system-on-chip. *IEEE Transactions on Biomedical Circuits and Systems*, 13(6):1736–1746, 2019.

[52] Yaoyao Jia, Ulkuhan Guler, Yen-Pang Lai, Yan Gong, Arthur Weber, Wen Li, and Maysam Ghovanloo. A trimodal wireless implantable neural interface system-on-chip. *IEEE Transactions on Biomedical Circuits and Systems*, 14(6):1207–1217, 2020.

[53] Shuenn-Yuh Lee, Chieh Tsou, Peng-Wei Huang, Po-Hao Cheng, Chi-Chung Liao, Zhan-Xien Liao, Hao-Yun Lee, Chou-Ching Lin, and Chia-Hsiang Hsieh. 22.7 a programmable wireless EEG monitoring SoC with open/closed-loop optogenetic and electrical stimulation for epilepsy control. In *2019 IEEE International Solid- State Circuits Conference - (ISSCC)*, pages 372–374, 2019.

[54] Jayant Charthad, Ting Chia Chang, Zhaokai Liu, Ahmed Sawaby, Marcus J. Weber, Sam Baker, Felicity Gore, Stephen A. Felt, and Amin Arbabian. A mm-sized wireless implantable device for electrical stimulation of peripheral nerves. *IEEE Transactions on Biomedical Circuits and Systems*, 12(2):257–270, 2018.

[55] Jihun Lee, Vincent Leung, Ah-Hyoung Lee, Jiannan Huang, Peter Asbeck, Patrick P. Mercier, Stephen Shellhammer, Lawrence Larson, Farah Laiwalla, and Arto Nurmikko. Neural recording and stimulation using wireless networks of microimplants. *Nature Electronics*, 4(8):604–614, Aug 2021.

[56] Hossein Kassiri, Muhammad Tariqus Salam, Mohammad Reza Pazhouhandeh, Nima Soltani, Jose Luis Perez Velazquez, Peter Carlen, and Roman Genov. Rail-to-rail-input dual-radio 64-channel closed-loop neurostimulator. *IEEE Journal of Solid-State Circuits*, 52(11):2793–2810, 2017.

[57] Ian Kuon and Jonathan Rose. Measuring the gap between FPGAs and ASICs. In *Proceedings of the 2006 ACM/SIGDA 14th international symposium on Field programmable gate arrays*, pages 21–30, 2006.

[58] Amara Amara, Frederic Amiel, and Thomas Ea. FPGA vs. ASIC for low power applications. *Microelectronics journal*, 37(8):669–677, 2006.

[59] Oscar W Savolainen, Zheng Zhang, Peilong Feng, and Timothy G Constandinou. Hardware-efficient compression of neural multi-unit activity. *IEEE Access*, 2022.

[60] Dongjin Seo, Ryan M. Neely, Konlin Shen, Utkarsh Singhal, Elad Alon, Jan M. Rabaey, Jose M. Carmena, and Michel M. Maharbiz. Wireless recording in the peripheral nervous system with ultrasonic neural dust. *Neuron*, 91(3):529–539, 2016.

[61] Gabriel Gagnon-Turcotte, Mehdi Noormohammadi Khiarak, Christian Ethier, Yves De Koninck, and Benoit Gosselin. A 0.13-μm cmos soc for simultaneous multichannel optogenetics and neural recording. *IEEE Journal of Solid-State Circuits*, 53(11):3087–3100, 2018.

[62] Peilong Feng, Michal Maslik, and Timothy G. Constandinou. Em-lens enhanced power transfer and multi-node data transmission for implantable medical devices. In *2019 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4, 2019.

[63] Eran Stark and Moshe Abeles. Predicting movement from multiunit activity. *Journal of Neuroscience*, 27(31):8387–8394, 2007.

[64] Robert D Flint, Eric W Lindberg, Luke R Jordan, Lee E Miller, and Marc W Slutzky. Accurate decoding of reaching movements from field potentials in the absence of spikes. *Journal of neural engineering*, 9(4):046006, 2012.

[65] Joseph E. O'Doherty, Mariana M. B. Cardoso, Joseph G. Makin, and Philip N. Sabes. Nonhuman primate reaching with multichannel sensorimotor cortex electrophysiology, May 2017.

[66] Joaquin Navajas, Deren Y Barsakcioglu, Amir Eftekhar, Andrew Jackson, Timothy G Constandinou, and Rodrigo Quian Quiroga. Minimum requirements for accurate and efficient real-time on-chip spike sorting. *J. Neurosci. Methods*, 230:51–64, 2014.

[67] Dong Han, Yuanjin Zheng, Ramamoorthy Rajkumar, Gavin Stewart Dawe, and Minkyu Je. A 0.45 v 100-channel neural-recording ic with sub-μw/channel consumption in 0.18μm cmos. *IEEE transactions on biomedical circuits and systems*, 7(6):735–746, 2013.

[68] Zheng Zhang, Oscar W Savolainen, and Timothy G Constandinou. Algorithm and hardware considerations for real-time neural signal on-implant processing. *Journal of Neural Engineering*, 19(1):016029, 2022.

[69] Matteo Pagin and Maurits Ortmanns. A neural data lossless compression scheme based on spatial and temporal prediction. In *2017 IEEE Biomedical Circuits and Systems Conference (BioCAS)*, pages 1–4. IEEE, 2017.

[70] David A Huffman. A method for the construction of minimum-redundancy codes. *Proc. IRE*, 40(9):1098–1101, 1952.

[71] Anton Biasizzo, Franc Novak, and Peter Korosec. A multi-alphabet arithmetic coding hardware implementation for small FPGA devices. *Journal of Electrical Engineering*, 64(1):44, 2013.

[72] György Buzsáki, Costas A Anastassiou, and Christof Koch. The origin of extracellular fields and currents—EEG, ECoG, LFP and spikes. *Nature reviews neuroscience*, 13(6):407–420, 2012.

[73] Oscar Herreras. Local field potentials: myths and misunderstandings. *Frontiers in neural circuits*, 10:101, 2016.

[74] Nir Even-Chen, Dante G Muratore, Sergey D Stavisky, Leigh R Hochberg, Jaimie M Henderson, Boris Murmann, and Krishna V Shenoy. Power-saving design opportunities for wireless intracortical brain–computer interfaces. *Nature Biomedical Engineering*, pages 1–13, 2020.

[75] Deren Y Barsakcioglu, Yan Liu, Pooja Bhunjun, Joaquin Navajas, Amir Eftekhar, Andrew Jackson, Rodrigo Quian Quiroga, and Timothy G Constandinou. An analogue front-end model for developing neural spike sorting systems. *IEEE transactions on biomedical circuits and systems*, 8(2):216–227, 2014.

[76] R Quian Quiroga, Zoltan Nadasdy, and Yoram Ben-Shaul. Unsupervised spike detection and sorting with wavelets and superparamagnetic clustering. *Neural computation*, 16(8):1661–1687, 2004.

[77] David M Brandman, Tommy Hosman, Jad Saab, Michael C Burkhart, Benjamin E Shanahan, John G Ciancibello, Anish A Sarma, Daniel J Milstein, Carlos E Vargas-Irwin, Brian Franco, et al. Rapid calibration of an intracortical brain–computer interface for people with tetraplegia. *Journal of neural engineering*, 15(2):026007, 2018.

[78] Katarzyna M Szostak, Laszlo Grand, and Timothy G Constandinou. Neural interfaces for intracortical recording: Requirements, fabrication methods, and characteristics. *Frontiers in Neuroscience*, 11:665, 2017.

[79] Song Luan, Ian Williams, Michal Maslik, Yan Liu, Felipe De Carvalho, Andrew Jackson, Rodrigo Quian Quiroga, and Timothy G Constandinou. Compact standalone platform for neural recording with real-time spike sorting and data logging. *Journal of neural engineering*, 15(4):046014, 2018.

[80] Jameson Thies and Amirhossein Alimohammad. Compact and low-power neural spike compression using undercomplete autoencoders. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 27(8):1529–1538, 2019.

[81] Tong Wu, Wenfeng Zhao, Edward Keefer, and Zhi Yang. Deep compressive autoencoder for action potential compression in large-scale neural recording. *Journal of neural engineering*, 15(6):066019, 2018.

[82] Sonia Todorova, Patrick Sadtler, Aaron Batista, Steven Chase, and Valérie Ventura. To sort or not to sort: the impact of spike-sorting on neural decoding performance. *Journal of neural engineering*, 11(5):056005, 2014.

[83] Eric M Trautmann, Sergey D Stavisky, Subhaneil Lahiri, Katherine C Ames, Matthew T Kaufman, Daniel J O'Shea, Saurabh Vyas, Xulu Sun, Stephen I Ryu, Surya Ganguli, et al. Accurate estimation of neural population dynamics without spike sorting. *Neuron*, 103(2):292–308, 2019.

[84] George W Fraser, Steven M Chase, Andrew Whitford, and Andrew B Schwartz. Control of a brain–computer interface without spike sorting. *Journal of neural engineering*, 6(5):055004, 2009.

[85] Elon Musk et al. An integrated brain-machine interface platform with thousands of channels. *Journal of medical Internet research*, 21(10):e16194, 2019.

[86] Reid Harrison, Paul Watkins, Ryan Kier, Robert Lovejoy, Daniel Black, Richard Normann, and Florian Solzbacher. A low-power integrated circuit for a wireless 100-electrode neural recording system. In *2006 IEEE International Solid State Circuits Conference-Digest of Technical Papers*, pages 2258–2267. IEEE, 2006.

[87] Beata Jarosiewicz, Anish A Sarma, Daniel Bacher, Nicolas Y Masse, John D Simeral, Brittany Sorice, Erin M Oakley, Christine Blabe, Chethan Pandarinath, Vikash Gilja, et al. Virtual typing by people with tetraplegia using a self-calibrating intracortical brain-computer interface. *Science translational medicine*, 7(313):313ra179–313ra179, 2015.

[88] Zheng Zhang and Timothy G Constandinou. Adaptive spike detection and hardware optimization towards autonomous, high-channel-count bmis. *Journal of Neuroscience Methods*, 354:109103, 2021.

[89] Wasim Q Malik, John P Donoghue, and Leigh R Hochberg. Real-time control of a clinical brain-computer interface using continuous wideband multiunit activity.

[90] Eric Drebitz, Bastian Schledde, Andreas K Kreiter, and Detlef Wegener. Optimizing the yield of multi-unit activity by including the entire spiking activity. *Frontiers in neuroscience*, 13:83, 2019.

[91] Joshua I Glaser, Ari S Benjamin, Raeed H Chowdhury, Matthew G Perich, Lee E Miller, and Konrad P Kording. Machine learning for neural decoding. *Eneuro*, 7(4), 2020.

[92] Christopher Olah. Understanding LSTM networks. 2015.

[93] Aditya Jain, Ramta Bansal, Avnish Kumar, and KD Singh. A comparative study of visual and auditory reaction times on the basis of gender and physical activity levels of medical first year students. *International Journal of Applied and Basic Medical Research*, 5(2):124, 2015.

[94] Dejan Markovic, Chen Chang, Brian Richards, Hayden So, Borivoje Nikolic, and Robert W Brodersen. Asic design and verification in an fpga environment. In *2007 IEEE Custom Integrated Circuits Conference*, pages 737–740. IEEE, 2007.

[95] Rajeev Jayaraman. Physical design for fpgas. In *Proceedings of the 2001 international symposium on Physical design*, pages 214–221, 2001.

[96] Thomas Brochier, Lyuba Zehl, Yaoyao Hao, Margaux Duret, Julia Sprenger, Michael Denker, Sonja Grün, and Alexa Riehle. Massively parallel recordings in macaque motor cortex during an instructed delayed reach-to-grasp task. *Scientific data*, 5(1):1–23, 2018.

[97] Thomas M Hall, Kianoush Nazarpour, and Andrew Jackson. Real-time estimation and biofeedback of single-neuron firing rates using local field potentials. *Nature Comms.*, 5:5462, 2014.

[98] Oscar W Savolainen and Timothy G Constandinou. Lossless compression of intracortical extracellular neural recordings using non-adaptive huffman encoding. In *2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC)*, pages 4318–4321. IEEE, 2020.

[99] Luis Fernando Nicolas-Alonso and Jaime Gomez-Gil. Brain computer interfaces, a review. *Sensors*, 12(2):1211–1279, 2012.

[100] Matteo Pagin, Florian Jetter, Günther Zeck, and Maurits Ortmanns. Lossless compression of neural signals with predictor schemes achieving more than fivefold data reduction of in-vitro recorded retinal signals.

[101] Ernest Jamro, Maciej Wielgosz, and Kazimierz Wiatr. FPGA implementation of the dynamic huffman encoder. In *Proc. IFAC Workshop on Programmable Devices and Embedded Systems, Brno*, pages 60–65, 2006.

[102] Oscar W Savolainen and Timothy G Constandinou. Investigating the effects of macaque primary motor cortex multi-unit activity binning period on behavioural decoding performance. *bioRxiv*, 2020.

[103] Oscar Savolainen, Zheng Zhang, Peilong Feng, and Timothy Constandinou. https://github.com/next-generation-neural-interfaces/hardware-efficient-mua-compression: Code, results and hardware designs for hardware-efficient compression of neural MUA, Feb 2022.

[104] Oscar Savolainen, Zheng Zhang, Peilong Feng, and Timothy Constandinou. Zenodo: Hardware-efficient compression of neural multi-unit activity - data and results; doi: 10.5281/zenodo.6265023, 2022.

[105] Zishen Xu, Wei Wu, Shawn S Winter, Max L Mehlman, William N Butler, Christine M Simmons, Ryan E Harvey, Laura E Berkowitz, Yang Chen, Jeffrey S Taube, et al. A comparison of neural decoding methods and population coding across thalamo-cortical head direction cells. *Frontiers in neural circuits*, 13:75, 2019.

[106] Mikhail A Lebedev. How to read neuron-dropping curves? *Frontiers in systems neuroscience*, 8:102, 2014.

[107] Oscar Savolainen, Zheng Zhang, and Tim. Constandinou. Code and results for 'ultra low power, event-driven data compression of multi-unit activity'. `https://github.com/Next-Generation-Neural-Interfaces/Comparison-of-ultra-low-power-hardware-efficient-MUA-compression-schemes`, 2022.

[108] Joseph G Makin, Joseph E O'Doherty, Mariana MB Cardoso, and Philip N Sabes. Superior arm-movement decoding from cortex with a new, unsupervised-learning algorithm. *Journal of neural engineering*, 15(2):026010, 2018.

Appendices

Appendix A

Dataset details

The datasets from Table 2.4 are detailed here.

A.1 Flint-Slutzky 2012

The first dataset is the Flint-Slutzky 2012 dataset [64]. It has 12 recordings taken across 5 days. Each recording consists of SUA data measured with a 96-channel Utah array in M1 in a non-human primate performing a free reaching task. The SUA data was obtained by taking broadband recording sampled at 30 kHz, highpass filtering them at 300 Hz, manually thresholding the resulting signal at an average of 5.2 standard deviations above the mean potential, and then manually sorting. In this work, MUA was derived from the SUA via collation of intra-channel spikes. In this work, the X and Y-axis cursor velocities, sampled at 100 Hz, were used as the observed behavioral data.

A.2 Brochier et al. 2018

The second dataset is the Brochier et al. 2018 dataset [96]. This consisted of two recordings, one obtained from each of two non-human primates (referred to as L and N) from a 96-channel Utah array in both the M1 and the dorsal (PMd - subject L) or ventral (PMv - subject N) premotor cortex. The recording from subject L was approximately 1.5 hours long from a single session, and the recording from subject N was approximately an hour long, also from a single session. The broadband data was first filtered with a 1^{st} order 0.3 Hz high pass filter (full-bandwidth mode) and a 3^{rd} order 7.5 kHz Butterworth low pass filter. The SUA was then obtained via manual offline thresholding and sorting, from broadband data sampled at 30 kHz. For subject L, prior to thresholding, the data was highpassed at 250 Hz. Subject L had some noisy spikes that were not removed in this work. For subject N, prior to thresholding, the data was passband filtered between 0.25 and 5 kHz. The behavioural data consisted of labelled actions. As these were not continuous measurements, the behavioral data from Brochier et al. was not analysed in this work so as to keep the BDP metric consistent.

A.3 O'Doherty Sabes lab 2017-2020 processed

The third dataset is the O'Doherty Sabes lab 2017 (updated in 2020) dataset [65]. Two adult male Rhesus macaque monkeys (Indy and Loco) were recorded performing a free-reaching task. The neural recordings consisted of SUA from Utah arrays implanted in M1 (and for some sessions also in S1). As such there were

either 96 or 192 channels per session, where some channels had no spiking activity and so were removed. 37 sessions spanning 10 months were recorded from Indy, and 10 sessions spanning a month from Loco. The hand and cursor X and Y positions were sampled at 250 Hz. The X and Y velocities were taken to be the observed behavioral data in this work, taken as the 1^{st} derivative of position.

A.4 O'Doherty Sabes lab 2017-2020 raw broadband

This dataset consists of the raw broadband recordings from the previous dataset [65]. Only the subject 'Indy' had broadband recordings.

The recordings are across 30 sessions. They are not segmented into trials, and have an average duration of ∼520 seconds. The broadband data was pre-amplified with a PZ2 Preamplifier (Tucker-DEDvis Technologies, Alachua, FL) with a frequency response of 3 dB: 0.35 Hz – 7.5 kHz; 6 dB: 0.2 Hz – 8.5 kHz [108]. There was also an anti-aliasing filter built-in to the pre-amplifier: 4^{th} order low-pass with a roll-off of 24 dB per octave at 7.5 kHz. It was then sampled at $F_s = 24414.0625$ Hz and 16-bit resolution using a RZ2 BioAmp Processor (Tucker-DEDvis Technologies, Alachua, FL). There is an anti-aliasing filter built-in to the recording amplifier: 2^{nd} order low-pass with a roll-off of 12 dB per octave at 7.5 kHz.

A.5 Hall and Jackson 2014

The data consists of raw Broadband microwire recordings from NHP M1 cortex, sampled at 16-bit resolution and 24.4 kHz [97]. The data came from 3 animals (Dusty, River, Silver) and a 'Ukiah' dataset, which was understood to originate from one of the 3 animals but using a different microwire array. The recording lengths belong to vector $RL = [205, 364, 396, 473, 646]$ s. Respectively, these values are the minimum length, the 1^{st} quartile, the median, the 3^{rd} quartile, and the maximum. The number of microwire electrodes ranged from 10 to 24, depending on subject.

Appendix B

Supplementary Material - Chapter 4: Static Huffman Compression of MUA

B.1 Data Compression analysis process, full details

In this section, we determined the effect of various compression architectures on BR and therefore communication power. The role of the number of encoders, S, BP, and histogram memory size, which is used in assignment of channels to encoders, were investigated. However, firstly, not all static Huffman encoders will be useful for compressing MUA data at a given BP. As such, to identify the subset of interesting Huffman encoders, a Machine-Learning (ML) strategy was adopted. This is because the subset of interesting Huffman encoders, to be implemented on-implant, should be found in a way that is compatible with real-life application. I.e., the choice of the subset of Huffman encoders implemented on-implant should not be influenced by the data to be measured on-implant, as this is not known prior to implantation.

An approach was taken where all possible static Huffman encoders of length S were considered. During each training-validation round, the set of Huffman encoders was reduced by removing the encoder that contributed the least to the compression. Eventually, only $u = 1$ encoder, i.e. the 'best encoder', was left, in which case no assignment or histogram were necessary and every channel used the same encoder. This is discussed further in the Supplemental Material, Section B.1.3.1.

As required for a machine learning approach, a training/validation split was used. The training data was used to reduce the set of Huffman encoders. The validation data was used to represent neural data recorded on-implant, on which the quality of the assignment and compression was measured.

The training and validation data, from A, were split by taking the full recording from a channel and assigning it to either training or validation. The training and validation channels were randomly selected with a 50/50 split. The channels from the Flint and Sabes datasets were split separately, so that both were represented 50/50 in training and validation. All 960 Flint channels in A were considered, and a random 2000 out of 4224 channels in the Sabes data in A were considered. The Sabes data was limited so that the contribution of the Flint data was not overshadowed, and so prevented overfit to the Sabes data. It warrants mentioning that this training-validation split is distinct from that in the Supplementary Material, Section 4, that looked at the effect of BP and S on BDP, although both splits concerned data from A.

B.1.1 Sorted Codeword Length Vector representation

We wanted to determine how the best u ($u \in Z, 1 \leq u \leq h$) Huffman encoders, from the set of all possible h Huffman encoders with S fields, would perform in compressing MUA data from multiple channels where each

Table B.1: Demonstrating a non-redundant representation of Huffman encoder codeword lengths, the SCLV. (a) All Huffman code combinations for S = 3; (b) Vectors of the lengths of the Huffman codes, with copies removed. (c) Sorted Codeword Length Vectors, with copies removed.

(a)

All Huffman Encoders of Length S = 3			
Huffman code combination	**1st CW***	**2nd CW**	**3rd CW**
1	0	10	11
2	0	11	10
3	00	1	01
4	01	00	1
5	11	0	10
6	00	01	1

***CW = Codeword**

(b)

Non-redundant Codeword Length Vectors, S = 3			
Huffman code length combination, key	**1st CW length**	**2nd CW length**	**3rd CW length**
1	1	2	2
2	2	1	2
3	2	2	1

(c)

Non-redundant Sorted Codeword Length Vector (SCLV)			
SCLV combination	**1st CW length**	**2nd CW length**	**3rd CW length**
1	1	2	2

channel was assigned to its ideal encoder. As such, all possible Huffman encoders of length S were produced. For example, for $S = 3$, all possible Huffman encoder codeword combination are given in Table B.1 (a).

There are many redundant configurations of Huffman encoders, assuming the order of the symbols in unimportant. The only constraint is that, in a Huffman encoder, the shortest codeword should represent the most likely symbol in the data, the 2nd shortest should match the 2nd most likely, etc. As such, the Huffman encoders were reduced down to a SCLV representation, which consisted of the ascending sorted vector of codeword lengths. This process is shown in Table B.1 (a-c). In the case of $S = 3$, it can also be observed in Table B.1 (c) that all $h = 6$ encoders reduce down to a single SCLV. Taking the dot product of the SCLV and sorted histogram, where the histogram is sorted in descending order, gives the length of the data, in bits, after compression by a Huffman encoder with the same SCLV as shown in Table B.2 (c). Dividing the dot product by the number of samples gives *avlen* from Eq. 2 in the main manuscript, where dividing further by BP gives the BR. This enables us to assign encoders to channels based on which encoder gives the smallest BR (Table B.2 (d)). It warrants mentioning that, to actually compress the data, a Huffman encoder that matches the SCLV is required. However the BR can be obtained from the SCLV via its dot product with the sorted histogram.

Based on the BDP vs. S results that will be shown later in this section, and our knowledge of the hardware costs of large S values that will be shown in Supp. Mat. Section B.5, we opted for the full integrated results to only examine S values between 2 and 9 for our set of BPs. This allowed relatively good BDP values, as discussed later, while minimising the resources. Therefore, for each integer value of $S\,\{S \in \mathbb{Z},\, 2 \leq S \leq 9\}$, the full set of SCLVs was produced. The details are given in the next section.

B.1.2 Producing the Full Set of Huffman Sorted Codeword Length Vectors

For each integer value of $S\,\{S \in \mathbb{Z},\, 2 \leq S \leq 9\}$, the full set of SCLVs was produced by creating every non-redundant combination of probability vectors p of length S, at a given discrete resolution q. Each probability

Table B.2: Taking the dot product between the SCLVs (a) and example Channel Sorted Histograms (b) to obtain the size of the compressed data blocks (c). From (c), we can determine the best SCLV-channel pairs. In (d), the channel-average length *avlen* in bits of the encoded symbols after each channel is compressed using its ideal encoder is obtained, with $L = 1000$.

(a)

SCLVs					
SCLV \Symbol	**0**	**1**	**2**	**3**	**4**
1	1	2	3	4	4
2	2	2	2	3	3
3	1	3	3	3	3

(b)

Channel Sorted Histograms (SH)				
Symbol \Chan.	**1**	**2**	**3**	**4**
0	650	500	300	950
1	200	200	300	5
2	100	120	200	31
3	50	100	100	7
4	0	80	100	7

(c)

Dot Product of Channel SHs and SCLVs				
SCLV \Chan.	**1**	**2**	**3**	**4**
1	**1550**	**1980**	2300	1109
2	2050	2180	**2200**	2014
3	1700	2000	2400	**1100**
Best SCLV	**1**	**1**	**2**	**3**

(d)

$$avlen = \frac{1}{n}\frac{\Sigma_{i=1}^{n} min(dotprod[:,i])}{L}$$

$$avlen = \frac{1}{4}\frac{1550 + 1980 + 2200 + 1100}{1000} = 1.7075 \text{ bits}$$

value $p[k]$, with $k\{k \in \mathbb{Z}, 1 \leq k \leq S\}$, was equal to $q \times j$, where q was the discrete increment and $j \in \mathbb{Z}^+$ so that $0 \leq p[k] = q \times j \leq 1$. q was set to a low value of 0.1, and each non-redundant probability vector p of length S was created.

Each p was then normalised to p' so all of its k elements summed to 1. A Huffman encoder was then trained on p', where p' was used to represented the frequencies of arbitrary symbols. Each Huffman encoder was then reduced down to the SCLV representation, and the non-redundant SCLVs for each S stored. With small enough q, every possible Huffman encoder, and so every possible SCLV, is generated.

B.1.3 ML process

B.1.3.1 Training

The system was trained by assigning the SCLVs to the training channels based on which SCLV gave the best BR for each channel. Multiple channels could be assigned the same SCLV, and all of the data in each channel was used.

To identify interesting SCLVs, i.e. train the system, the following strategy was used. Firstly, each training channel's sorted histogram was obtained. Secondly, one by one, with replacement, each SCLV was removed. The average BR across channels was then measured after each channel was assigned to its best remaining SCLV, e.g. as in Table B.2. Channels could be assigned to any SCLV, except the missing one. After the total BR had been obtained for each missing SCLV, the SCLV that was found to be the least effective SCLV for compressing the training data, in concert with the other SCLVs, was removed. As such, at the end of the training round, one SCLV had been removed. This process is represented in Fig. B.1 (b).

B.1.3.2 Validation

Between each training round, validation was performed. The purpose of validation was to measure the BR when each 'on-implant' channel was assigned to its ideal encoder from the selection of encoders available

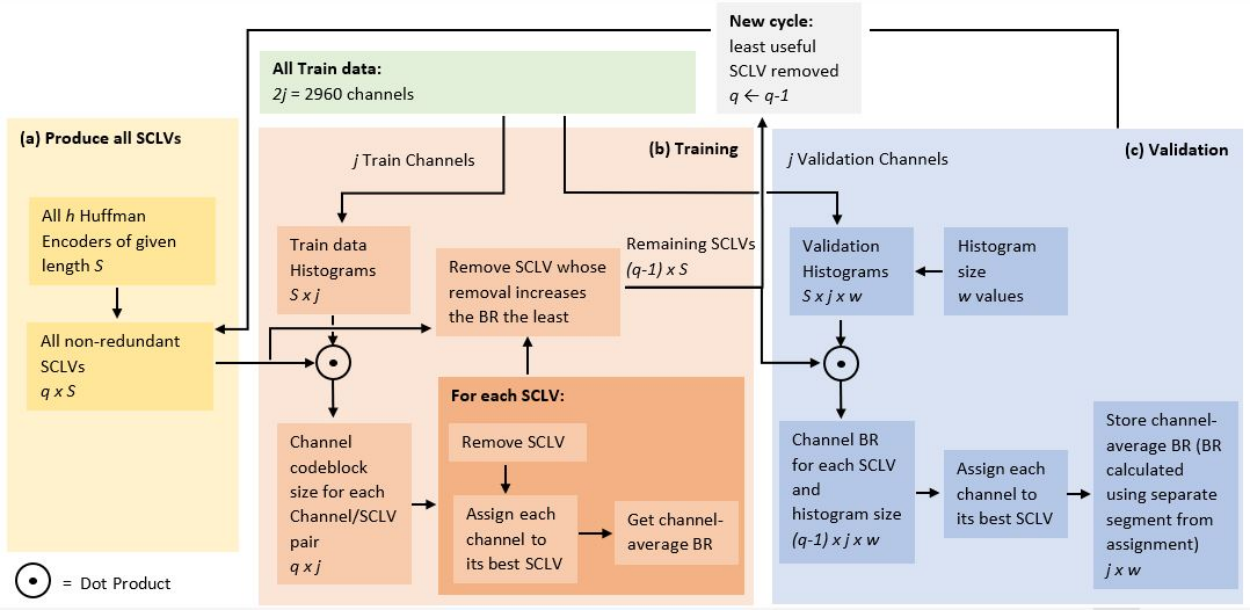


Figure B.1: Encoder-selection machine learning algorithm. Italics represent data dimensions. First, all possible SCLVs are produced (as shown in (a)). Then, each round selects an increasingly smaller subset of SCLVs based on which give the best compression in the training set (as represented in (b)). At each round, i.e. with different numbers of SCLVs, the compression is tested on the validation dataset for different on-implant histogram sizes (as represented in (c)).

during the current training round. In other words, it was tested what the on-implant BR would be given the selection of encoders found during that training round. The method of assigning on-implant MUA channels to encoders should be realistic to implement in hardware. In this work, the assignment was done by using a segment from the beginning of each on-implant channel recording to produce a sample histogram. Each channel was then assigned to an encoder based on which encoder gave its histogram the smallest BR. This is represented in Fig. B.1 (c).

How much of the beginning of the 'on-implant' validation recording was used to calculate the sample histogram depended on the size of the histogram. The considered histogram sizes were $S \times 2^d - 1$ samples, where $d \in [2, 3, 4, ..., 9, 10]$. Each histogram bin, of which there were S, was given a size of $2^d - 1$ maximum samples, i.e. of d bits. Once $2^d - 1$ samples had been measured across all bins, the histogram growth ended, the histogram was sorted, and the channel was assigned to an encoder.

After assignment, the resulting BR was calculated. This was done for each channel by taking the rest of the validation data, that which had not been used for assignment, and calculating its BR after compression. In particular, only a segment of the remaining data was used, where the segment was always equal in length to half of the total recording length. This was so that the amount of compressed data was the same across all histogram sizes. This is shown in Fig. B.2, where the beginning of the recording is used for assignment, and a segment of the rest for calculating the BR. The average BR across validation channels was then stored for each histogram size.

We then moved to the next training-validation round. At each round, the least useful SCLV was removed based on the training data. The validation channels were then assigned to the remaining SCLVs, and the average BR for each histogram memory size stored. The process continued until only 1 encoder (i.e. the best encoder for the training data) was left, where all validation channels were automatically assigned to the last encoder. As such, for each round (i.e. number of non-redundant SCLVs), the mean BR, ideal set

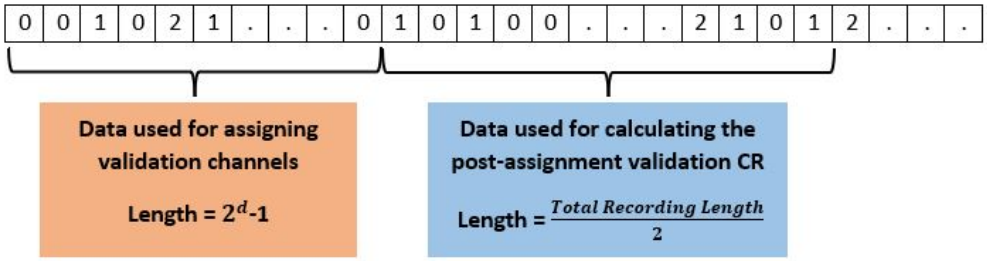


Figure B.2: Single-channel split of assignment (for sample histogram) and to-be-compressed validation data, based on histogram memory size d and total recording length.

of encoders, and effect of histogram size on BR were all determined. 30 different cross-validation iterations of this process were run in parallel, with different training-validation channel splits, and the results were combined.

It is important to mention that, for any channel, the compressed on-implant data should be sorted the same way its histogram was sorted. For example, if a firing rate of 3 MUA events per bin is found to be the most common firing rate in the sample histogram, the occurrence of 3 MUA events in a bin in the remaining to-be-compressed data should be given the shortest codeword during compression. This is because the sorted ordering in the compressed data has to be determined somehow, and so is approximated using the sample histogram for each channel. As such, the histogram serves to not only get an idea of the shape of the channel's histogram, used to find the ideal encoder for compression, but also to find the sorting order of the to-be-compressed data, as the most common firing rates should be assigned the shortest codewords within each encoder. We refer to this assignment of to-be-compressed firing rates to encoder fields, based on the sample histogram, as mapping (e.g. mapping the symbols to the encoder fields). As such, the size of the histogram affects the quality of both the assignment and the mapping, both which impact the BR. The sorting and mapping implementation is discussed further in Section B.4.

B.2 Parameter optimisation during determining the effect of S and BP on BDP

The only data that was observed was the training data A specified in Section 2.1.2 of the main manuscript. Each recording in A was split into training and testing, using a 90-10% split. The training data was then split into sub-training and validation using 5-fold cross-validation. The Wiener Cascade Filter hyper-parameters and the pre-processing parameters were then optimised on the folds. The optimised decoder hyper-parameters included the Wiener Cascade Filter degree, the alpha parameter, and the number of taps i.e. timesteps used in the Wiener Cascade Filter. l_2 (i.e. ridge or Tikhonov) regularisation was used by default. The optimised pre-processing parameters included the lag between the neural and behavioral data, and the width of the causal moving average window used to smooth the neural data. The optimised parameter values are given in Table B.3.

The best-performing parameters across the 5 folds for each S and BP were then used as the parameters for the decoder tested on the testing data, giving the BDP as a function of S and BP. Here, for testing the effect of S on BDP, BPs of 1, 5, 10, 20, 50 and 100 ms were investigated.

B.3 BDP Results

B.3.1 BDP as a function of BP and S - Train data A

From Fig. B.5, it warrants mentioning that BDP increases with BP, assuming the effect of increasing S has saturated to that of $S = max(x)$ (where the line stabilizes horizontally for each BP). However, the value at which the effect of S saturates increases for each BP. Interestingly, a lower BP can have a better BDP than a higher BP at the same S, since no information is being saturated for the lower BP. However, even low S values perform quite well relative to the peak values, even for a BP of 100 ms. It is also interesting that these results are opposite to those in [102], where BDP decreased linearly as a function of BP. However, [102] used LSTM decoders whereas this work uses Wiener Cascade Filters. It isn't surprising that deep learning algorithms may be able to find more information because at higher temporal resolution than simple decoders like the Wiener Cascade Filter. Also, [102] looked at the average effect of BP across different signal processing parameters, whereas this work looked at the effects when using fully optimised system parameters. It also warrants mention that, even if one generally gains BDP at a higher BP, one loses on temporal resolution of the decoded output. Furthermore, one can always bin the data to a higher BP prior to decoding if one wishes to gain a better BDP, as higher BPs are simply a lossy representations of lower BPs.

B.3.2 BDP as a function of BP and S - Test data B

From Fig. B.8, it can be seen that the BP/S relationship to BDP holds much the same in the test data B. However, the BDP values are generally significantly lower in the training data (Fig. B.5). The same parameter optimisation was performed, and so either the range of optimised parameters is non-ideal, the test recordings have less behavioral data in them, or they are more noisy.

B.4 FPGA realisation

The work in this section, involving hardware design, was carried out by Z.Z., who also authored this section.

We simulated the presented architectures on an FPGA target, Lattice ice40LP, in order to assess the overhead on power and resources brought by compression so as to guide our configuration selection. Such a low-power, high-performance FPGA with 40nm technology and small BGA package is ideal for the thinnest devices like implantable BMIs. All programs are written in Verilog, simulated on Mentor Modelsim Lattice Edition and synthesised with iCECube 2020.12.

Lattice ice40LP1K is an ultra-low-power FPGA board with 1280 logic cells and sixteen 4kbit memory blocks (bRAMs). The architecture of the FPGA implementation is shown in Fig. B.9, including the Binner, Histogram counter, Sorter, Mapper, Encoder selector, Encoders and Memory. Referring to Table 4.1 and Fig. 4.2 in the main manuscript, different configurations can be achieved by bypassing some of the components. In the 'one encoder' version, there is only one encoder in Encoders and therefore no need for the Encoder selector module. In the 'Without Mapping' configuration, we assume the events in the histogram are already sorted and the sorter and mapper are bypassed. The *Spike rate freq* is connected with the *Sorted freq* in the encoder selector, and the *New spike rate* is connected with the *Mapped spike rate* in Encoders. Similarly, in the 'Only Binning' version, only the Binner module is implemented. A detailed breakdown of Fig. B.9 is given below.

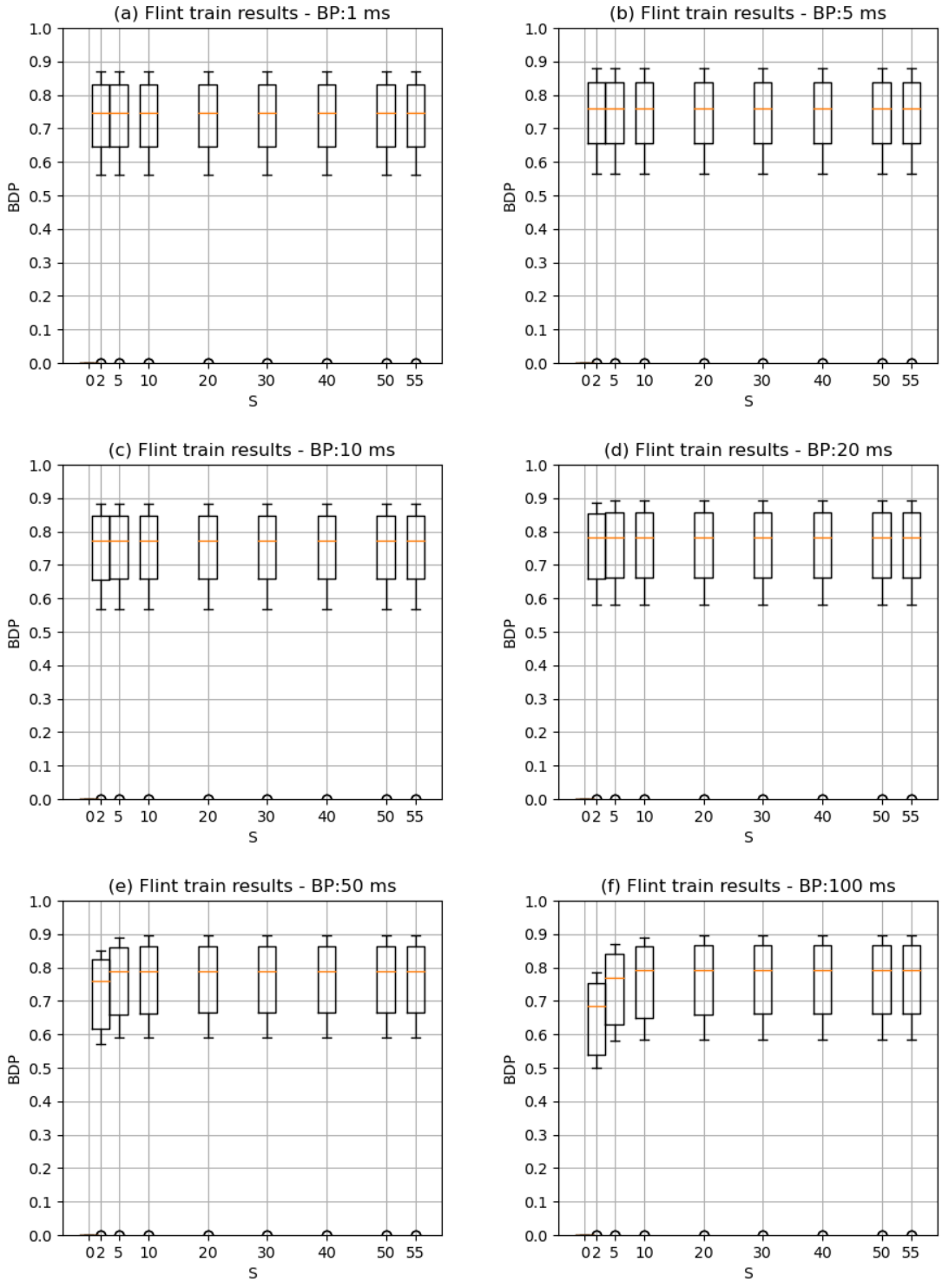


Figure B.3: Behavioral decoding performance (BDP) as a function of BP and S for the Flint data in A. Each S/BP combination was parameter optimised on 5-fold CV.

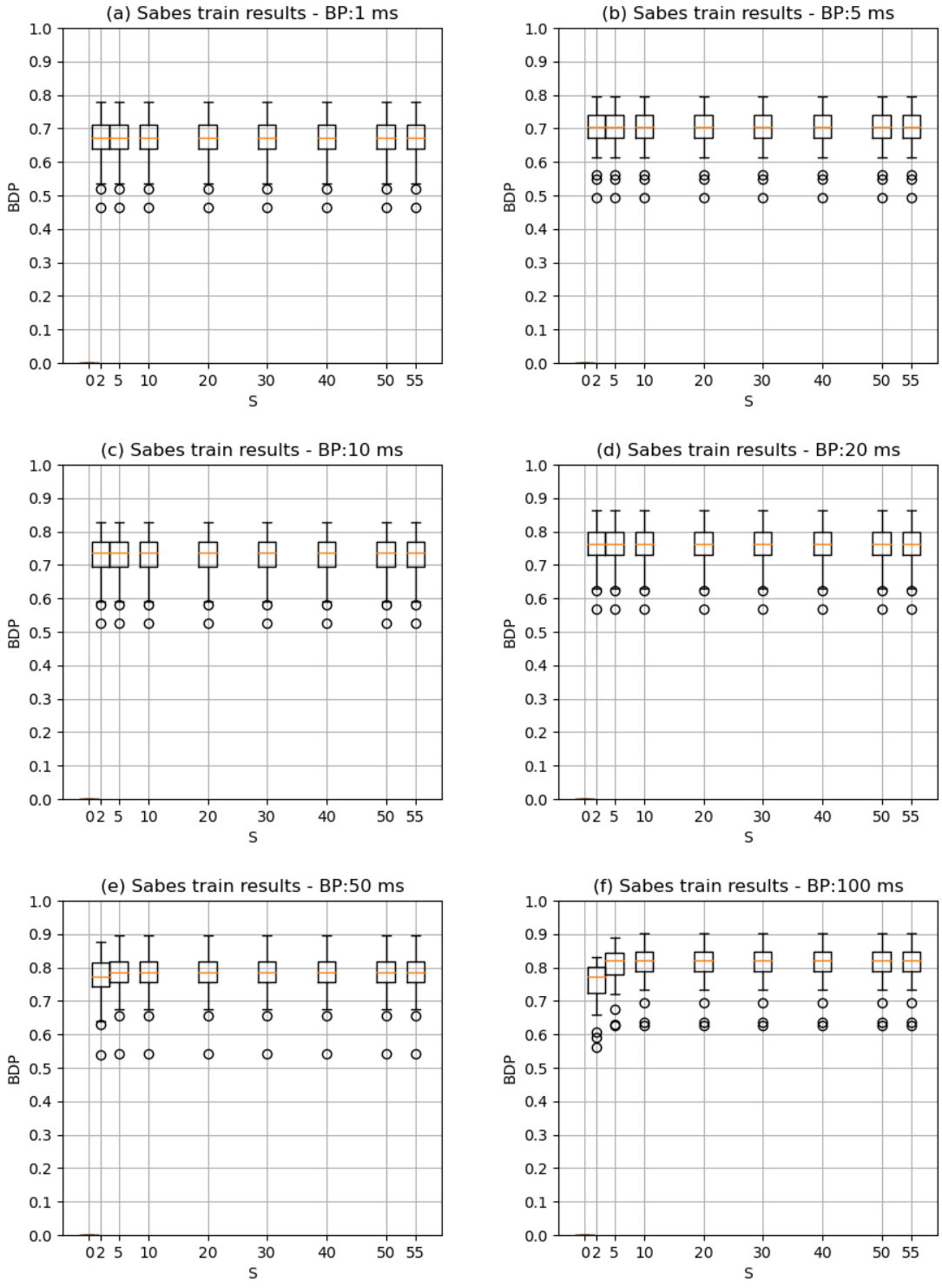


Figure B.4: Behavioral decoding performance (BDP) as a function of BP and S for the Sabes data in A. Each S/BP combination was parameter optimised on 5-fold CV.

Table B.3: Optimised parameters and hyper-parameters for the behavioral decoders.

Timesteps	5, 10, 15
Kinematic to neural lag (s)	0, 0.02, 0.04
Moving average window length (s)	0, 0.05, 0.1, 0.2
Alpha	0, 1e-4, 1e-2
CWT Degree	2, 3, 4

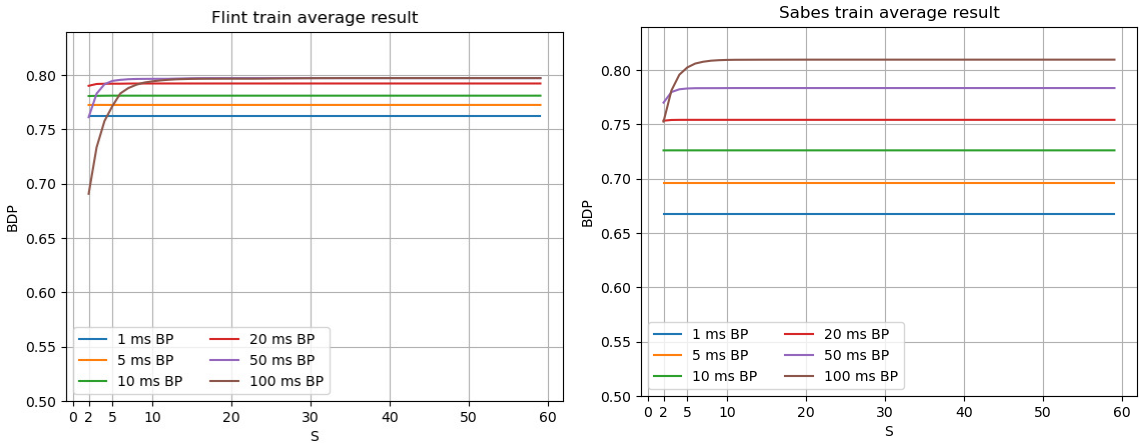


Figure B.5: Behavioral decoding performance (BDP) for Flint and Sabes training data as a function of BP and S, averaged across the recordings.

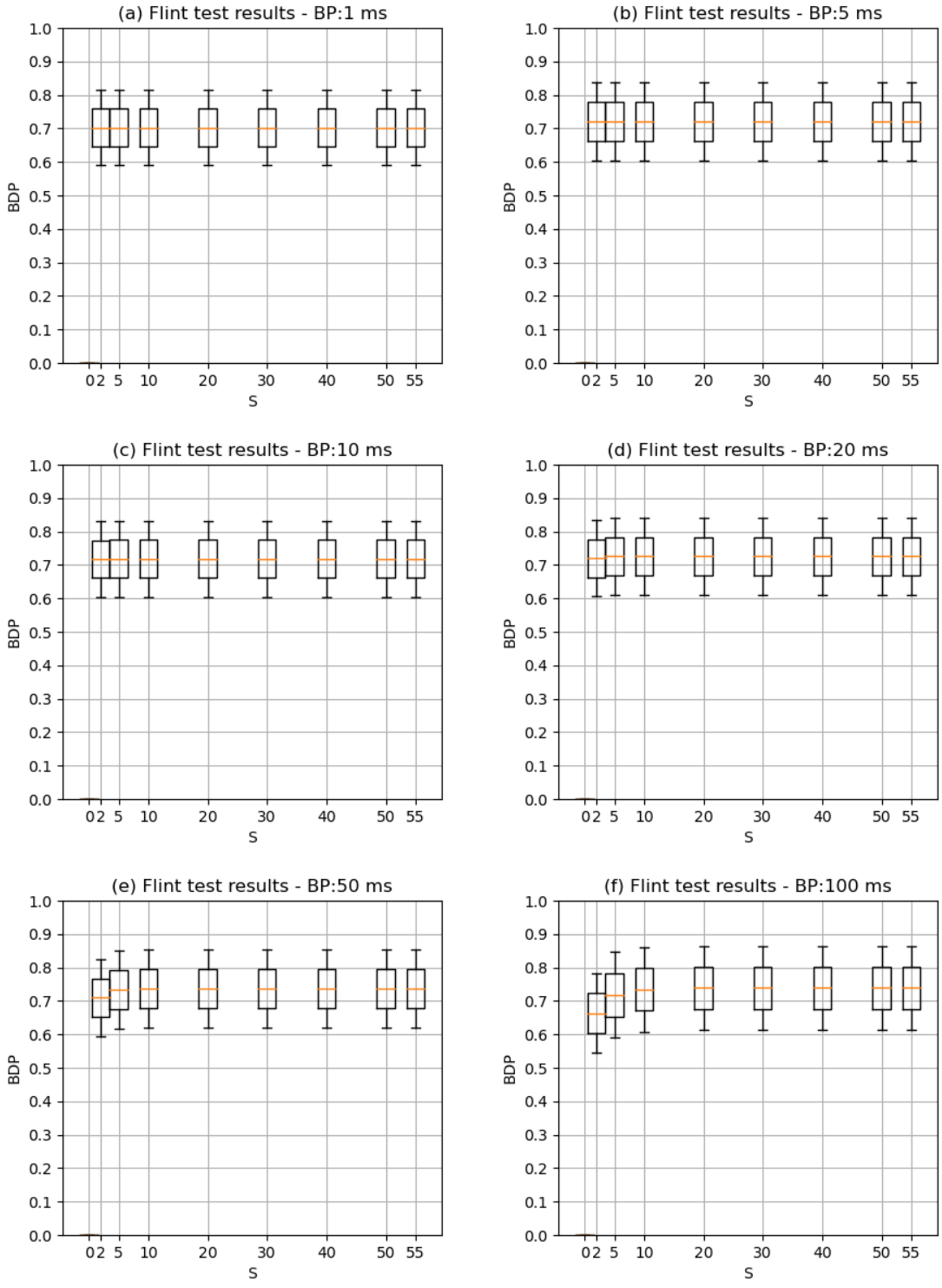


Figure B.6: Behavioral decoding performance (BDP) as a function of BP and S for the Flint data in B. Each S/BP combination was parameter optimised on 5-fold CV.

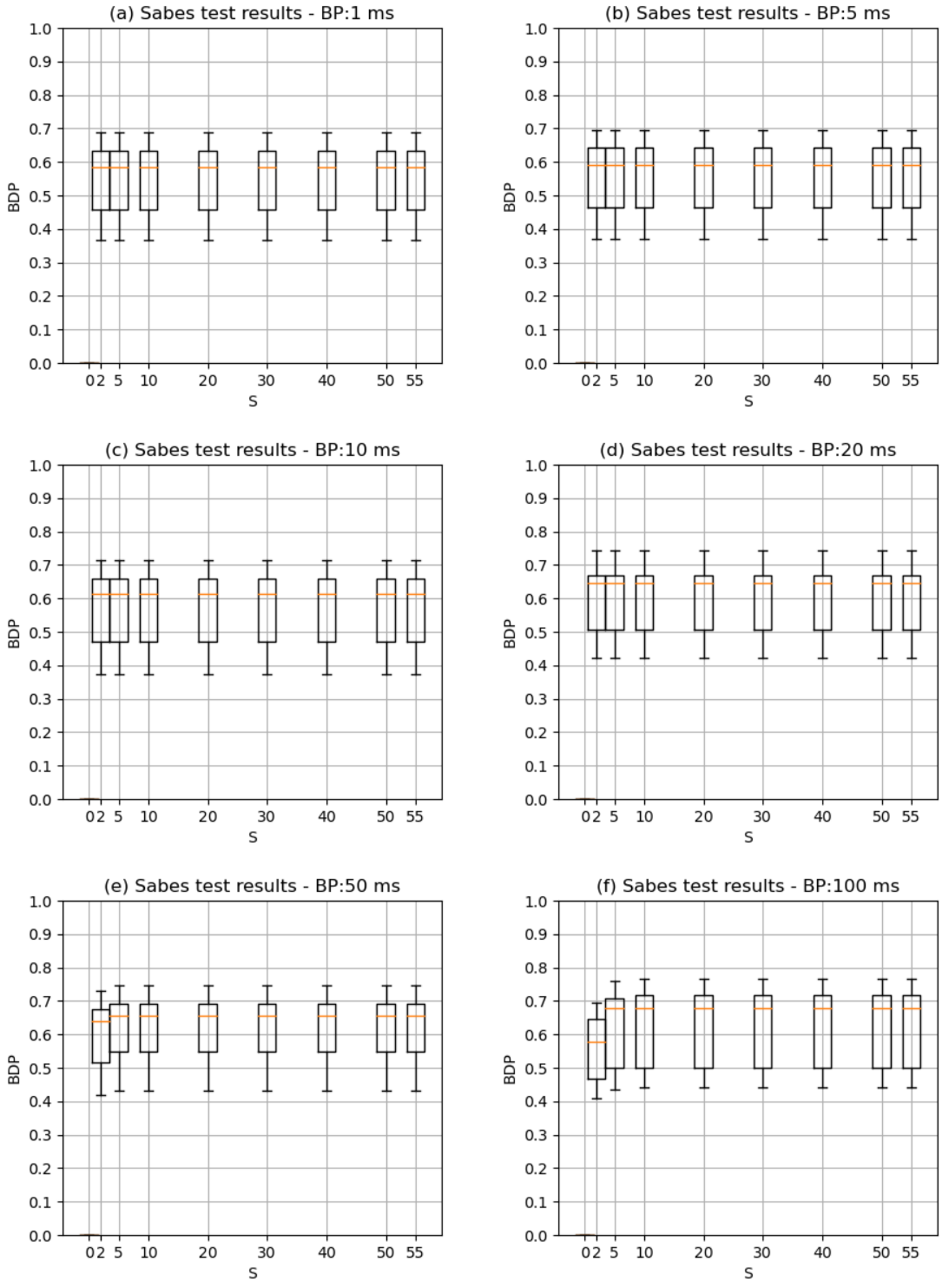


Figure B.7: Behavioral decoding performance (BDP) as a function of BP and S for the Sabes data in B. Each S/BP combination was parameter optimised on 5-fold CV.

B.4.1 Binner

A timer and a recurrent spike rate counter is used for the Binner, which counts the number of in-coming spike numbers in a given BP. The multiplexer (MUX) after the spike rate counter is used to clip the spike rate within the preset range and reset the spike rates stored in memory when a BP is over.

B.4.2 Histogram

The Histogram is implemented using a Finite State Machine. The state transition diagram is shown in Fig. B.9 (B). It accumulates the number of MUA events according to the *Bin finished* and new spike rate signals. When the histogram overflows, a finish signal is issued and the histogram is emptied for the next channel. As the maximum spike rate is clipped at $S-1$, the number of registers required for storing the different MUA frequencies is highly reduced. The index of the maximum value in the MUA histogram is recorded in the histogram counter, to be used in the Sorter.

B.4.3 Sorter and mapper

Sorting the spike rate frequency in order can be resource-hungry or time-consuming in hardware. The resources used for implementing a sorting algorithm such as merge sort or quick sort can overwhelm the whole system. For sorting MUA histograms, we can take advantage of the fact that the MUA histogram, even if it does not follow a decaying exponential, is almost always expected to follow a unimodal peak distribution. As such, the sorted histogram can be easily estimated by setting the index 0 at the index of the histogram maximum, and the index number will increment by iterating on both sides of the histogram peak. For example, values to the left of the peak will take odd index numbers of 1, 3, 5, etc., while values to the right of the peak will take even numbers. When indices can no longer be assigned on one side, the rest are assigned serially to the other side. An example is illustrated in Fig. B.10 A. Using such an estimation, we can reduce both the sorting space and time complexity to $O(n)$.

From a hardware perspective, this estimated sort algorithm can be implemented with a finite state machine (FSM). However, in Fig. B.9, we show a combinatorial implementation. As the estimated sort order is only affected by the most frequent spike rate index, we can easily create a LUT that defines the sorting order based on the measured maximum index. The spike rate Mapper can also be implemented using an identical LUT. A demo of the sorted indices LUT when $S=5$ is given in Fig. B.10. We also implemented two other sort algorithms: a swapping sort and estimation sort with sequential logic, which are given in Supplemental Material Section 5.

B.4.4 Encoder Selector

The encoder selector assigns encoders to channels. For the given channel, it selects the encoder that gives the minimum encoded data length, determined by taking the dot product of the sorted histogram with the SCLVs. The SCLVs are stored on-implant for each encoder. Since multiplications are resource-hungry in hardware, we replace the multiplications with bit shifts. The number of bits to shift for different encoders and codewords are stored on board.

B.4.5 Encoders

A Huffman encoder, in hardware, is a big LUT. Multiple encoders have been implemented on board. Different encoders are selected according to the encoder selector. The selected set of Huffman codewords and codeword length are then set as the outputs of this channel.

B.4.6 Memory unit

The Memory unit consists firstly of a channel counter that counts the processed channels to schedule the data flow among channels. Secondly it consists of RAM that stores each channels' current MUA FR, their sample histogram largest value's index during calibration (used for mapping during regular encoding operation) and their assigned encoders. Instead of using registers for each channel, utilising RAM increases the scalability, making it possible to upscale to thousands of channels within only hundreds of logic cells. However, the clock speed needs to be doubled to maintain the same data throughput. As the resources are highly constrained in lattice iCE40LP, the RAM implementation is preferred over using registers.

Fig. B.9 (c) shows the timing of the clocks. CLK is the system clock and a clock generator is used to generate MCLK for memory and PCLK for different processing units. The detection signal will be loaded at the negedge of the PCLK. The RAM will provide the stored parameters for different processing units at MCLK posedge. All processing happens on PCLK posedge and the results are stored back at the MCLK negedge. These modules work together to compress the MUA data of each channel. The histogram counter, sorter and encoder selector are used at the start of implant operation for encoder selection as a calibration process. During the calibration, for each channel, the sample histogram largest value's index and encoder assignment will be determined and stored into RAM. One can also periodically recalibrate each channel when the brain environment changes, or at some defined interval. After the calibration, the spike detection signal of different channels will flow through the binner, mapper, encoder and RAM interchangeably for compression.

B.4.7 Hardware implementation of the sort algorithms

Utilising the nature of the spike rate, we can customise sort operations to reduce the hardware cost. Besides the combinatorial implementation introduced in the main context. We here proposed other two different sort algorithms, we named it a real-time swap sort and the other is an estimated sort with sequential logic.

B.4.7.1 Swapping sort

The real-time swap sort the spike rate count histogram and also the order of spike rate (count lags) in real-time during histogram accumulation. It is therefore balances the computation load temporally. Based on the fact that every bin period the histogram counter reads the output spike rate from the binner, the spike histogram count can only be increased by 1. Therefore, swapping the updated spike rate count with the last value smaller than it can ensure the spike count histogram is in order.

In one iteration, the updated value only compare itself with the spike rate counts equal to the value before it is updated. The complexity is still $O(n)$ theoretically if the spike rate is random. However, as the actually spike rate count is expected to follow a unimodal peak distribution, the number of comparisons is much lower than $O(n)$ in practice.

B.4.7.2 Estimation sort with sequential logic

Three important registers and two register banks have been used: *Max* stores the most frequent firing rate, which is obtained in histogram count; *Start* stores the current location of the left of the most frequency firing rate being sorted; *End* stores the current location on the right of the most frequency firing rate being sorted; *Freq* is a bank of S registers that store the estimated sorted firing rate frequencies; *Rate* a bank of S registers that store the original lags (firing rate) of the frequencies before sorting.

The sorter stays at *IDLE* when counting the firing rate histogram. After counting finishing, the sort enters *LOAD* state, in which it will load the frequency of each firing rate from histogram into *Freq*, stores

Table B.4: Logic cells were used in two different settings for three sort implementations.

	Swapping sort	Estimation sort - Seq	Estimation sort - Comb
S = 5, hist size = 4	480	239	75
S = 9, hist size = 6	682	355	260

0 to S-1 in to *Rate* and set the current *Start* and *End* indices according to *Max*. When *Max* is 0, *Start* = 0. When *Max* is S - 1 (the largest possible firing rate), *Start* = S - 1, *End* = S. In other cases, *Start* = *Max* - 1, *End* = *Max* + 1. It will also store *Max* and its frequency at the first index of *Rate* and *Max*. If neither *End* = S nor *Start* = 0, the sorter will enter the *SORT* state, where *Start* and *End* will be stored in the next even index and odd index in *Rate*, the frequencies will also be stored in *Freq* accordingly. After that, *Start* will be reduced by one and *End* will be increased by one. If *End* = S, the sorter will enter *RSRT* state. In that state, the right side of the *Max* has been sorted, and only the left side, i.e. *Start* side will be processed. When *Start* becomes zero, sorting is finished. Note that, if *Start* is decreased to zero (the left side sort finishing), before *End* is increased S, the sorter will also enter *IDLE* state, as the rest on the right is already sorted.

Such an estimated sort, in the worst case when the maximum is the last value in the histogram, will take S clocks to get the result. However, the maximum is normally only appeared in the left half and it takes less than S/2 clocks after histogram count.

B.5 Hardware results

The work in this section, involving hardware design, was carried out by Z.Z., who also authored this section.

B.5.1 Resource usage

Resources usage reflects the area occupation of the implementation. The full results are given in Supplemental Material 2 as an excel spreadsheet, and the same is available on the Github at [103]. The histogram and encoder selector are two resource-hungry modules. The amount of resources required by the histogram is mostly dependent on increased histogram size, but the S values also have a significant but smaller impact. The resource usage of the encoder selector can exceed that of the histogram when S is larger than 7 because of the dot product (implemented with bit shifting) between two vectors with length S. Alternatively, all possible multiplication results could be pre-calculated and stored in a RAM. The selector logic would be simplified, however this would sacrifice the processing speed as the results would need to be fetched from RAM one by one and summed together. As a result, the calibration time would be increased. Such an alternative approach could be useful if the limitation of the resources is extreme or the configuration requires a large S, histogram size or number of encoders. It would be less effective when these values are small. Aiming at finding the most compact compression scheme, we opted to use the bit-shift implementation discussed in the Sup. Mat., Section B.4.

For the sorter, we have compared three different approaches: Swapping sort, and estimation sort with sequential and combinatorial implementations. A summary of the resources needed for the three implementations is given in Table B.4.

One can notice that compared to the Swapping sort, the sequential estimation sort reduced the required resources by half, which makes it possible to do on-implant sorting with limited resources available. More

noticeable, when $S = 5$, histogram size = 4, the resources of the combinatorial implementation is only one-third of the sequential implementation, making the sorting no longer the bottleneck for resource usage. This advantage is lessened when the settings are extreme. However, even when $S = 9$ and histogram size = 6, in which case the whole system can require too many logic cells to be implemented within our resources budget, the combinatorial one still uses fewer logic cells than the sequential version.

The remaining two modules, i.e. the Binner and Encoders, use few resources. These are normally below 100 LUTs+FFs each.

To guide the configuration selection, using only one encoder without sorting would be preferred because it can get rid of the histogram, sorter, mapper and encoder selector. If one has a histogram, increasing S would be preferred over increasing histogram size because increased histogram size has a large effect on both the selector and histogram counter, which are the two resource-dominating modules. These findings are purely from the resource perspective, the selection should also be guided jointly on the resultant total power and BDP.

B.5.2 Power consumption

Power consumption is another aspect of concern. We should guarantee that the added processing power does not exceed the reduced communication power. However, the estimated power indicates that the processing power consumption per channel is consistent among different configurations. As the histogram counter, sorter and encoder selector are only used during calibration, the Binner, Mapper, RAM and Encoder continuously consume energy. The binner and RAM tick at the processing clock/memory clock speed, but the input of the encoders only changes at $\frac{1}{BP}$ Hz, which is much lower than the clock speed. Therefore the power of the Binner and RAM dominate the FPGA dynamic power, which is around $0.96\,\mu$W per channel. The power of the encoder is negligible at 1 to 20 nW, the binner consumes about $0.46\,\mu$W per channel and RAM shares the remaining $0.5\,\mu$W per channel. For the remainder of this work, the combined compression/processing and communication power is referred to as the dynamic power. The board static power is $162\,\mu$W.

As all configurations use the binner and RAM, the processing power of different architectures is similar whether we encode the firing rate or not. Therefore the total power reduction we gain from the compression is proportional to the BR reduction.

Bases on the exploration of resources and power, we can conclude that it is the resources that constrain the algorithm complexity for the on-implant Huffman encoding.

B.6 Fixed Length vs. Variable Length Codewords and Bit-Flip Errors

An advantage of the multiplexed representation of MUA data is that the channel ID does not have to be explicitely encoded. This is because the channel ID is implicitely encoded in bit position. E.g.,

$$c = ceiling(k/m)$$

where $k \in [1 \leq k \leq n \times m]$ is the bit position and $c \in [1 \leq c \leq n]$ is the channel ID.

The principle technique of lossless compression is to transform fixed length codewords into variable length codewords. This is to take advantage of the potentially narrow and skewed nature of the data's histogram. However, fixed-length codewords have an advantage when it come to bit-flip errors during communication of multiplexed data. With fixed-length codes in multiplexed MUA, the channel ID is implicitly encoded in bit position, and this remains fixed over time. Therefore, any bit-flip error will only affect the communicated

number of MUA events recorded on one channel, in the time period of question associated with the code block. No other channels will be affected.

With variable-length codewords, such as those produced by lossless compression techniques, all of the symbols after the codeword with the bit-flip error may be affected. Specifically, it may offset the relationship between symbols, encoders and channels. If different channels use different encoders, this can quickly make the entire sequence after the corrupted bit undecodable. For example in Table 2.2 (d), if the 3rd bit in the encoded multiplexed signal flips to a 1, then the sequence will be decoded as:

```
3, 2
```

instead of:

```
3, 0, 1
```

The consequences of the bit flip will propagate downstream. In this case, the 3rd symbol will be decoded as whatever comes next in the sequence, which should have corresponded to the 4th symbol. Given different channel-encoder pairings, the entire sequence post-error will likely become corrupted.

As such, if the bit-flip error rate is sufficiently high, some method may be required to reduce the consequences of bit flip errors. The first obvious candidate is to reduce the BP, where the BR is increased, but the temporal resolution of the data is increased. Therefore, if a data packet is corrupted, the user will not notice much difference as the next data packet will arrive shortly. The second clear candidate is noisy channel encoding.

B.6.1 Noisy channel encoding

Noisy channel encoding involves adding parity bits to communicated data blocks. These parity bits have some relationship to the data. If a bit-flip error occurs, comparison of the codeblock and the parity bits can signal the existence of errors. The parity bits can even signal the locations of the errors, depending on the thoroughness of the noisy channel encoding and the number of parity bits.

If noisy channel encoding is used, the compression derived from variable-length codewords vs. fixed-length codewordsneeds to be sufficient to warrant the addition of parity bits to the code block.

In this work, noisy channel encoding was not explicitly applied as the rate of bit flip errors is unknown. However, calculating the required number of parity bits for each codeblock length is simple given the bit flip error rate and code block length.

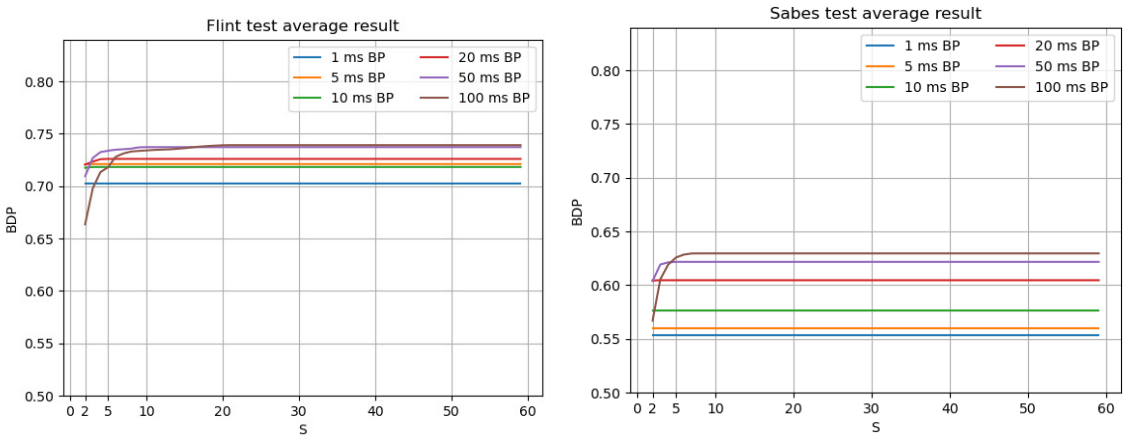


Figure B.8: Behavioral decoding performance (BDP) for Flint and Sabes testing data as a function of BP and S, averaged across the recordings.

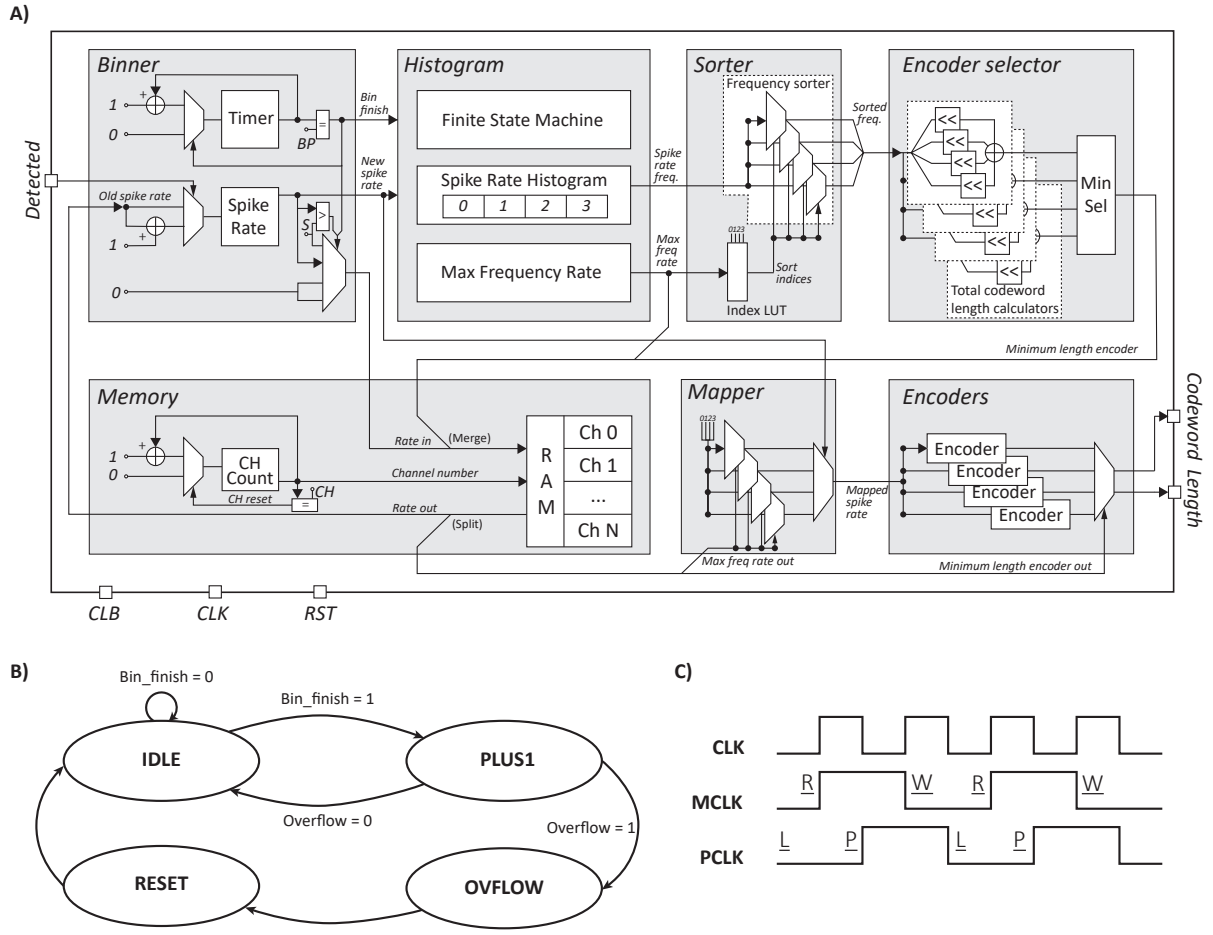
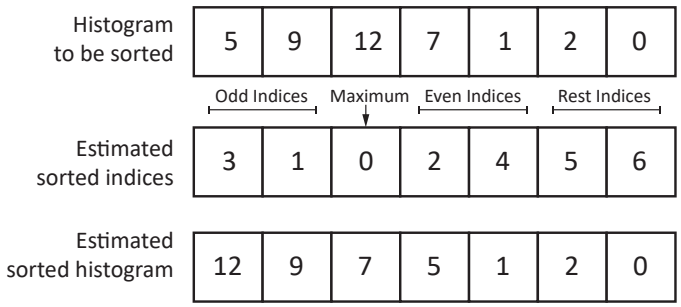


Figure B.9: A). The FPGA implementation includes Binner, Histogram, Sorter, Mapper, Encoder selector, Encoders and Memory. For conciseness, several circuits are not shown: Clock generator for memory clock and processing clock, reset and re-calibration logic, clip for *New spike rate* and MUXs selecting *Minimum length encoder out*, *Sorted indices out* as the input to the RAM after calibration. B). State transition diagram of the finate state machine in Histogram module. C). Clock timing diagram of the system clock (CLK), Memory clock (MCLK) and Processing clock (PCLK). MCLK drives the RAM, it is read at the posedge (R) and written at the negedge (W). PCLK trigers the Binner and Histogram at its posedge (P). It also synchronise the detected signal at its negedge (L). The update of channel number is also happened at the negedge of PCLK (L).

A)



B)

Max/Index	0	1	2	3	4
0	0	1	2	3	4
1	1	0	2	3	4
2	3	1	0	2	4
3	4	3	1	0	2
4	4	3	2	1	0

Figure B.10: A). A demo for the estimated sort algorithm. Index 0 of the sorted histogram will be the maximum of the unsorted histogram. Odd indices will be placed on the left while the even indices will be placed on the right. The rest of the indices will be placed serially on the unsorted side. B). A demo of the sorted indices when $S = 5$. Max indicates the index of the peak value in the histogram, and Index is the histogram field. Based on the peak value, the indices in the table are given to the histogram fields.

Appendix C

Supplemental Material - Chapter 5: Comparison to Event-Driven Architectures for Compressing MUA

C.1 Calculating Bit Rates for Each Encoding

Here we give detailed explanations of each encoding, as well as how their Bit Rates (BR) are calculated. The BR notation is $\mathrm{BR_{X-y}}$, where X is the encoding version, e.g. Basic (B), Static Huffman (SH), Adaptive Huffman (AH) or Entropic Bandwidth (E), and where y is the encoding, e.g. Windowed (W), EED, DED, or GED.

C.1.1 Windowed encoding

C.1.1.1 Static Huffman

Firstly, the average probability distribution of MUA FRs across channels, $\bar{p_i}$, was calculated. It is given by:

$$\bar{p_i} = \frac{1}{n}\sum_{j=1}^{n} p_{j,i} \tag{C.1}$$

where $p_{i,j}$ is the measured probability of a FR of i MUA events/bin occurring on channel j in a given BP across all of the considered data. As each channel's histogram had the same number of samples, $\bar{p_i}$ is proportional to the sum of the channels' histograms, meaning it is representative of the multi-channel data. $p_{i,j}$ was calculated for each considered number of channels n, BP and S, using a random selection of channels.

As such, the average BR per channel is given by:

$$\mathrm{BR_{SH-w}} = \frac{\sum_{i=0}^{S-1}(CLVs - she_i \times \bar{p_i})}{n \times BP} \quad \text{[bps/channel]} \tag{C.2}$$

where $CLVs - she_i$ is the codeword length for FR $= i$ events/bin, where the codeword is given by the static Huffman encoder trained on a decaying exponential probability vector of length S, where FRs of 0 to $S-1$ events/bin are encoded.

C.1.1.2 Adaptive Huffman

For the Adaptive Huffman version, a single Huffman encoder was trained on $\bar{p_i}$. Using $\bar{p_i}$ meant that each channel shared the same encoder. As such, it differs from the previous implementation in that the encoder is trained on the to-be-compressed data, whereas the previous encoder was trained on a decaying exponential, assumed to be a good fit to the data. The resulting BR is given by:

$$\mathrm{BR_{AH-s}} = \frac{\sum_{i=0}^{S-1}(CLVs_i \times \bar{p_i})}{n \times BP} \quad \text{[bps/channel]} \quad \text{(C.3)}$$

where $CLVs_i$ is the i^{th} element of the codeword length vector for the windowed encoding, which is a vector of the Huffman code lengths where the i^{th} element corresponds to the value with probability $\bar{p_i}$.

C.1.1.3 Entropic Bandwidth

The minimum number of bits per channel per second required for the windowed encoding is given by the average of the channels' Shannon entropies [15] divided by the BP:

$$\mathrm{BR_{E-s}} = -\frac{1}{n \times \mathrm{BP}} \sum_{j=1}^{n} \sum_{i=0}^{S-1} p_{j,i} \times log_2\left(p_{j,i}\right) \quad \text{[bps/channel]} \quad \text{(C.4)}$$

Here we assume each channel j has its own encoder with codeword lengths $-log_2\left(p_{j,i}\right)$, giving the best compression possible for this encoding.

C.1.2 Explicit asynchronous encoding

C.1.2.1 Basic implementation

In the windowed architecture, the FR per channel is encoded. The channel ID is implicitly encoded in bit position. To the best of the authors' knowledge, the following asynchronous architectures are proposed for the first time in this work. In these asynchronous architectures, the channel ID is explicitly encoded. E.g., in this 'explicit asynchronous' architecture, the channel ID is only sent out if a non-zero FR of i events/bin occurs on that channel, followed by a binary codeword representing i. If a channel has a FR of 0, that channel is missing entirely from the communicated codeblock. As such, although the channel ID has to be explicitly encoded, this is only if the FR is not 0 for that channel in the bin. For sparse signals, i.e. where FRs not equal to 0 are rare, this may offer improved compression over the windowed paradigm, where even FRs of 0 need to be communicated for each channel. As with the windowed architecture, the number of encoded FRs can be saturated at an integer value S.

As the channel IDs need to be explicitly encoded, the simplest method is to give the channels a standard binary codeword of length k_1, where:

$$k_1 = ceil(log_2(n)) \quad \text{[bits]} \quad \text{(C.5)}$$

Similarly, the MUA FR for each channel is encoded as a binary codeword of length m_2:

$$m_2 = ceil(log_2(S-1)) \quad \text{[bits]} \quad \text{(C.6)}$$

This differs from m' in that a codeword of length m' can only encode up to $2^{m'}-1$ FRs, whereas a codeword of length m_2 can encode up to 2^{m_2} FRs. This is because the case of FR = 0 does not need to be encoded for the explicit asynchronous encoding. An exception is if $S = 2$, i.e. $m_2 = 1$ and i is limited to 0 and 1. In

which case the m_2-length vector is unnecessary as the occurrence of the non-zero FR = 1 is already encoded in the sending out of the channel ID.

An example of the basic implementation, with $n = 4$, $k_1 = 2$ and $m_2 = 2$, is given by:

0001 0100 1011

This shows that channel 1 (00) had a FR of 2 events (01) in the bin, channel 2 (01) had FR = 1 (00), channel 3 (10) had FR = 4 (11), and channel 4 had a FR = 0 (absent). In this work, this encoding was named the explicit asynchronous encoding because the FR per channel is explicitly encoded in the m_2-length codeword that follows the channel ID.

For calculating the BR, one only sends out the channel ID and FR codeword if a non-zero FR occurs on that channel in the given bin. As such, one sends out the channel ID of channel j and its m_2-length codeword with probability pc_j:

$$pc_j = 1 - p_{j,0} \tag{C.7}$$

where $p_{j,0}$ is the probability of FR = 0 on channel j in the average time bin at the given BP. As such, the BR is given by:

$$\mathrm{BR}_{\mathrm{B-EED}} = \frac{1}{n \times \mathrm{BP}} \sum_{j=1}^{n} pc_j \left(k_1 + m_2\right) \qquad \text{[bps/channel]} \tag{C.8}$$

C.1.2.2 Static Huffman implementation

In this implementation for the explicit asynchronous method, we used a SH encoder to losslessly compress the FRs. Only the FRs are compressed, whereas the channel IDs are not. This is very simple to implement in hardware while likely giving good compression. It uses the same k_1 length codeword for the channel IDs, but uses varying length codewords for the MUA FRs. As in the SH windowed implementation, the FR encoder was trained on a decaying exponential.

The BR is given by:

$$\mathrm{BR}_{\mathrm{SH-EED}} = \frac{1}{n \times \mathrm{BP}} \sum_{j=1}^{n} \left(pc_j \times k_1 + \sum_{i=1}^{S-1} (CLVshe_i \times p_{j,i}) \right) \quad \text{[bps/channel]} \tag{C.9}$$

where $CLVshe_i$ is the codeword length of FR = i, given by the static Huffman encoder trained on a decaying exponential probability vector of length $S-1$, where FRs of 1 to $S-1$ events/bin are encoded. Importantly, there is no FR codeword for $i = 0$ in this encoding, and the shortest codeword length is $CLVshe_1$.

C.1.2.3 Adaptive Huffman implementation

In this AH implementation, we seek to compress both the FRs and the channel IDs, using the correct probability distributions observed on-implant. As such, this architecture differs from the previous one in that both the FR and the channel IDs are losslessly compressed. The prefix code nature of Huffman encoding can be taken advantage of to have two separate encoders: the first encoding the channel IDs, the second encoding the FRs. This is possible since we know the encoded FR will always follow the encoded channel ID, and so we know which encoder to use at each codeword. A design choice was made to train the FR encoder based on the average channel histogram, rather than have a separate encoder per channel. This is much more hardware efficient.

The ideal codeword length assigned to a FR depends on the relative probability of sending out FRs. As such, the FR encoder is trained on the relative probabilities $\widehat{pe_i}$:

$$\widehat{pe_i} = \frac{\sum_{j=1}^{n} p_{i,j}}{\sum_{i=1}^{S-1}\left(\sum_{j=1}^{n} p_{i,j}\right)},\ i > 0 \tag{C.10}$$

where $\widehat{pe_i}$ represents the relative probability of any $i > 0$ FR occurring relative to other $i > 0$ FRs, averaged across all channels. It is similar to $\bar{p_i}$, the difference being that the case of $i = 0$ is discounted.

The use of relative probabilities to train encoders warrants spending some time on, because the motivation for absolute vs. relative probabilities is something that sets the considered asynchronous architectures apart from the windowed one.

Relative probability gives how likely each symbol is to show up relative to other symbols, which is what encoders are trained on so as to give the ideal codeword length to each symbol. However, this becomes strange when it is no longer guaranteed that each channel will output a symbol that will receive a codeword. In this case, FRs = 0 are not encoded. To give the ideal codeword length, we should not encode the case where the codeword length will always be zero. Additionally, the sum of probabilities used to train the Huffman encoder should sum to 1. As such, we train the encoder on the relative probability of each non-zero FR occurring, so as to not waste bits on codewords that will never be used, e.g. FR = 0.

However, ultimately, the probability of that FR occurring is unchanged. Regardless of its optimal codeword length, it will occur at the frequency it occurs at. As such, we need to differentiate between the absolute probability, which predicts how often the FR will occur, and the relative probability, which gives the ideal codeword length for the FR. And so, for the average FRs across channels, we differentiate between $\bar{p_i}$ and $\widehat{pe_i}$.

The same concept applies to the channel IDs. Some channels may be dysfunctional and always have FRs of 0, in which case they should not be encoded. While unlikely, it is ideal in terms of compression to account for it. The relative probability $\widehat{pc_j}$ of sending out channel j's ID is given by:

$$\widehat{pc_j} = \frac{pc_j}{\sum_{l=1}^{n}(pc_l)} \tag{C.11}$$

The BR is then given by:

$$\mathrm{BR}_{\mathrm{AH-EED}} = \frac{\frac{1}{n}\sum_{j=1}^{n}(CLVc_j \times pc_j) + \sum_{i=1}^{S-1}(CLVfr_i \times \bar{p_i})}{BP}\ [\text{bps/channel}] \tag{C.12}$$

where $CLVc_j$ is the Huffman codeword length of the j^{th} channel ID, and $CLVfr_i$ is the Huffman codeword length for the i events/bin MUA FR, trained on $\widehat{pe_i}$. The right hand side concerning $CLVfr$ does not need to be divided by n because the channels have already been averaged in $\bar{p_i}$.

C.1.2.4 Entropic Bandwidth

To calculate the lower limit of how many bits are needed for the explicit asynchronous encoding, four aspects are calculated. The first two are the absolute and relative probability of channels having FR > 0, i.e. pc_j and $\widehat{pc_j}$ as calculated in Eq. C.7 and C.11.

The second two are the absolute and relative probabilities of each FR per channel, $p_{i,j}$ and $\widehat{p_{i,j}}$. $\widehat{p_{i,j}}$ is different from $p_{i,j}$ in that the case of a $i = 0$ FR is not represented. As such, $\widehat{p_{i,j}}$ is given by:

$$\widehat{p_{i,j}} = \frac{p_{i,j}}{\sum_{i=1}^{S-1} p_{i,j}}, i > 0 \tag{C.13}$$

In the entropy calculation, each channel was considered to have its own unique encoder, which is ideal in terms of compression if not hardware realisation. This is why we used $\widehat{p_{i,j}}$ instead of $\widehat{pe_i}$, the latter of which is averaged across channels.

From these four elements, we calculated the entropic BR, which is the minimum average number of bits per channel per second required to represent the channel IDs and the subsequent $i > 0$ FRs.

$$\mathrm{BR}_{\mathrm{E-EED}} = -\frac{1}{n \times \mathrm{BP}} \sum_{j=1}^{n} \left(pc_j \times log_2(\widehat{pc_j}) + \sum_{i=1}^{S-1} (p_{i,j} \times log_2(\widehat{p_{i,j}}) \right) [\text{bps/channel}] \quad \text{(C.14)}$$

C.1.3 Delta-asynchronous encoding

C.1.3.1 Basic implementation

It may be that, rather than communicating out discrete codewords for the channel IDs, it would be better to send out the encoded difference between adjacent channel IDs. In effect, we are delta-sampling the communicated out channel IDs, i.e using a run-length encoding. If a channel has a FR above 0, it is given a Δ value by subtracting the ID of the previous channel to have a FR above 0, $j_{previous}$, from the current channel's ID, $j_{current}$. I.e., $\Delta = j_{current} - j_{previous}$. The first channel ID to be sent out in the BP will be Δ-sampled relative to a channel ID of 0.

However, there is no clear basic implementation of this method that does not involve compressing the delta-samples. This is because the amount of bits required to represent each delta-sampled channel ID $\Delta \in [\mathbb{Z}, 1 \leq \Delta \leq n]$ is the same k_1 bits as required to represent the channel IDs directly. This is because the most extreme case of $\Delta = n$, where only the last channel has a FR above 0, needs to be accounted for. Therefore $ceil(log_2(n)) = ceil(log_2(n)) = k_1$ bits are required to encode Δ. As such, the basic implementation of this method was ignored and we moved straight the SH implementation.

C.1.3.2 Static Huffman implementation

In this implementation, we assigned a SH codeword to the Δ-sampled channel IDs, where smaller Δ values received shorter codewords. This way, if channel IDs are communicated out often, the Δ codewords will likely be shorter than the standard k_1 bits.

C.1.3.2.1 Training the Δ encoder

However, a design choice has to be made on how to train the delta-sampled channel ID encoder. It was decided to train the decoder on a decaying exponential so that smaller Δ values were given shorter codewords. However, the ideal rate of decay is an unknown parameter. Larger rates of decay will give smaller differences shorter codewords, which is ideal if channel IDs are communicated out often. Smaller rates of decay will give more equal codeword lengths for different Δ values. This is ideal if theΔ values vary significantly. As such, a set of $d = -10^f$ rates were considered, where d is the exponent of a decaying exponential and f is an integer with $f \in [\mathbb{Z}, -3 \leq f \leq 2]$. As such, decaying exponents of -10^{-2} to -10^3 were considered. The SH encoder for delta-sampled channel IDs was then trained on a probability vector $p_d(d, \Delta)$ of length n, where:

$$p_d(d, \Delta) = \frac{e^{d \times \Delta}}{\sum_{g=1}^{n-1} e^{d \times g}} \quad \text{(C.15)}$$

As shown in Eq. C.15, $p_d(d, \Delta)$ was normalised across all Δ so as to sum to 1 for each d value. For each BP, S and n combination, each considered decay exponent d was used to train a delta-sampled channel ID static encoder. The channel IDs were then delta-sampled. Δ was then encoded using the static encoder trained on $p_d(d, \Delta)$. The BR was stored for each d, and the best performing d was then chosen for each BP, S and n combination.

C.1.3.2.2 Saturating at a Δ_{max} value

However, having an on-implant Huffman encoder stored in a Look-up Table (LUT), with a unique codeword for each Δ value for large n, can have significant memory requirements. As such, it was analysed whether a shorter codebook of Δ values could be used, truncated at a Δ_{max} value. If a $\Delta > \Delta_{max}$, then a reset signal, equal to $\Delta_{max} + 1$, was communicated out along with how many times the reset signal should be read and the remainder, $\Delta_{max} - \Delta$. As such, Δ was transformed into a concatenated sequence of:

$$\Delta \rightarrow \Delta_{max} + 1,\, \gamma,\, B$$

where $\Delta_{max} + 1$ is the reset signal, γ is an integer value expressing how many times we should read the reset signal, and B is the remainder. For example, for $\Delta_{max} = 10$ and $\Delta = 45$:

$$\Delta \rightarrow 11, 4, 5$$

where we first communicate the reset signal indicating that Δ_{max} has been exceeded ($\Delta_{max} + 1 = 11$), then we communicate how many times it has been exceeded ($\gamma = 4$), and finally the remainder ($B = 5$).

The $\Delta_{max} + 1$ reset value was encoded using the delta-encoder trained on $p_d(d, \Delta)$, and the quotient value γ and remainder B were given fixed-length codewords of lengths θ_γ and θ_B respectively. Since the fixed-length codewords only ever occur after the reset signal, the sequence is fully decodable.

This saved memory, since there were fewer possible Δ codewords. However, limiting Δ to Δ_{max} in this way increased the processing hardware resources and power. Δ_{max} values of $\Delta \in [2^{[3,\,6,\,7,\,8,\,9,\,10]},\, n]$ were considered. If for a given parameter combination, $n < \Delta_{max}$, that parameter combination was ignored. The values γ and B are given in the Table C.1.

C.1.3.3 Adaptive Huffman implementation

For the AH version, no Δ_{max} was considered. Similar to the SH version, for the AH version the data was iterated through time-wise and the Δ values stored across all timesteps and channels. In effect, we counted the number of occurrences of each Δ value, and stored them in a vector $u\Delta$. A probability vector $\widehat{p_\Delta}$, of length n, was then obtained that gave the relative probabilities of each Δ value occurring throughout the data:

$$\widehat{p_\Delta} = \frac{u\Delta}{\sum_{\Delta=1}^{n} u\Delta} \tag{C.16}$$

$\widehat{p_\Delta}$ was used to train a Huffman encoder, that gave the encoded codeword lengths CLV_Δ for each Δ value. The BR is given by:

$$\mathrm{BR}_{\mathrm{AH-DED}} = \frac{\sum_{\Delta=1}^{n}(CLV_\Delta \times u\Delta)}{n \times v/\mathrm{BP}} + \frac{\sum_{i=1}^{S-1}(CLVfr_i \times \bar{p}_i)}{\mathrm{BP}} \quad \text{[bps/channel]} \tag{C.17}$$

Table C.1: Values for γ, B and θ values for Δ_{max} in the SH Delta-asynchronous encoding. 'floor' is the floor operation, and 'mod' is the modulus.

Parameter	**Value**
γ	floor((Δ - 1) / Δ_{max})
B	mod(Δ - 1, Δ_{max}) + 1
θ_γ [bits]	ceil($\log_2$((n / Δ_{max}) - 1))
θ_B [bits]	ceil($\log_2$(Δ_{max}))

where $CLV\Delta_i$ is the codeword length of the Δ values given by the Huffman encoder trained on $\widehat{p\Delta}$. v is the data length in seconds, i.e. $v = 100\,\text{s}$.

On the left hand side of Eq. C.17, we can see that we take the codeword length weighted sum of all of the occurring delta values, and divide by the total data length in samples, i.e. $n \times 100\,\text{s/BP}$, to get the average bps/channel for the channel IDs. We combine this with the FR component on the right hand side, which is the same as in Eq. C.12, where we train a single AH encoder for all of the channels.

C.1.3.4 Entropic Bandwidth

There is a minor point to mention in the entropic calculation for the explicit asynchronous method with delta-sampled channel IDs. We assume that, unlike in the AH version above, each channel has its own unique FR decoder. This is optimal for compression but not for hardware complexity. This is theoretically possible as the channel ID is encoded as delta-samples, and from the delta-sample the actual channel ID is derived off-implant. Having derived the channel ID off-implant, one can decode the channel's FR using the channel's FR decoder.

The BR is given by:

$$\text{BR}_{\text{E-DED}} = -\frac{\sum_{\Delta=1}^{n}(u\Delta \times log_2(\widehat{p\Delta}))}{n \times v/\text{BP}} - \frac{\sum_{j=1}^{n}\sum_{i=1}^{S-1}(p_{i,j} \times log_2(\widehat{p_{i,j}}))}{\text{BP}}\text{[bps/chan]} \quad \text{(C.18)}$$

C.1.4 Group Event-Driven encoding

C.1.4.1 Basic implementation

In this GED encoding, one uses position and stop symbols to encode the FRs. As in the explicit asynchronous encoding, one explicitly encodes the channel ID, but here one encodes the FR per channel implicitly in channel ID position. For example, in decimal,

$$2\,4\,\textit{stop}\,1\,6\,\textit{stop}\,\textit{stop}\,3$$

signifies that channels 2 and 4 had a FR of $i = 1$ in the given bin, channels 1 and 6 had a FR of $i = 2$, channel 3 had FR $i = 4$, and the rest of the channels had FR $i = 0$. As with the explicit asynchronous encoding, this asynchronous encoding benefits from not having to encode channels with FRs of 0 in the given bin, making it perform well for sparse signals. As with the other architectures, the measurable FRs can be saturated at S.

There is a design choice to be made on whether each number $i \geq 2$ of events should have its own stop codeword, or whether all event numbers should share the same stop codeword. We opted to have all of the transitions have their own codeword. This alleviates the risk of requiring multiple stop codewords next to each other. This risk is shown in the example above where no channels had 3 events/bin but one had 4. As such, all values to the right of the i stop symbol have FRs $\geq i$, where $i \geq 2$. E.g., channel IDs before the $i = 2$ stop symbol have FRs equal to 1, and there is no $i = 1$ stop symbol.

The stop symbol for each $i \geq 2$ has, in the average codeblock, a probability ps_i of occurring. ps_i is equal to the probability that there will be at least 1 channel on which there is a FR of i. This is equal to the complement of the probability of no channels having a FR of i. As such, ps_i, a vector of length $S - 2$, is given by:

$$ps_i = 1 - \prod_{j=1}^{n}(1 - p_{j,i}), \quad i \geq 2 \quad \text{(C.19)}$$

For the codeword lengths, the simplest implementation is to give the stop symbols and the channel IDs a length of k_2 bits, where:

$$k_2 = ceil(log_2(n + S - 2)) \quad \text{[bits]} \quad \text{(C.20)}$$

E.g. $n = 2$, $S = 4$, means that there are 2 channels, FRs between 0 and 3 inclusive can be encoded, and $k_2 = 2$. Channel 1 gets a codeword of 00, channel 2 gets a codeword 01, and the stop symbols for $i = 2$ and $i = 3$ get codewords of 10 and 11 respectively. It warrants mentioning that for large numbers of channels, typical $S \leq 20$ values will have relatively little impact [59]. As such, relatively large dynamic ranges may be achievable will little to no cost in this basic GED architecture for large n. Alternatively, the number of channels should be chosen carefully in conjunction with S so as to maximize the use of the range given by k_2.

The average BR per channel is given by:

$$\text{BR}_{\text{B-ga}} = \frac{1}{n \times \text{BP}} \left(\sum_{j=1}^{n} pc_j \times k_2 + \sum_{i=2}^{S-1} ps_i \times k_2 \right) \quad \text{[bps/channel]} \quad \text{(C.21)}$$

C.1.4.2 Static Huffman implementation

The GED encoding does not have an obvious SH version. This is because the codeword lengths are all relative to both channel ID and stop symbol frequency. While these could be estimated, e.g. the channels all have an equal likelihood of occurring and the stop symbols follow a decaying exponential, it becomes a lot of guesswork very quickly in terms of the relative probabilities. As such, no SH version was considered for the GED encoding. The basic version, where all codewords had an equal length, was assumed to be close to what a SH version would have achieved.

C.1.4.3 Adaptive Huffman implementation

In the explicit asynchronous architecture, one knows that the channel ID codeword is followed by a codeword encoding the channel's FR. However, in this architecture it is unknown whether the next codeword will be a channel ID or a stop symbol. As such, the codewords need to uniquely decodable, and so must come from the same encoder. This is also why the channel IDs and stop symbols shared the same binary k_2 codeword basis in the basic implementation. The Huffman encoder in the GED architecture must be trained on the combined probabilities of the channel ID and stop symbol occurring.

$$pg = [pc, ps] \quad \text{(C.22)}$$

where pg is the collated probability vector of pc and ps. ps_0 and ps_1 are not included in the collation as they are not valid values (see Equation C.19). The relative probabilities are given by:

$$\widehat{pg_a} = \frac{pg_a}{\sum_{a=1}^{n+S-2} pg_a} \quad \text{(C.23)}$$

where $a \in [\mathbb{Z}, 1 \leq a \leq n + S - 2]$ is the index of the concatenated probability vector. The encoder was trained on $\widehat{pg}$. The BR is given by:

$$\text{BR}_{\text{AH-ga}} = \frac{\sum_{a=1}^{n+S-2} (CLVg_a \times pg_a)}{n \times \text{BP}} \quad \text{[bps/channel]} \quad \text{(C.24)}$$

where $CLVg_a$ is the length of the Huffman codeword associated with the a^{th} element of the combined probability vector pg. Our convention was that if $a \leq n$, $CLVg_a$ represents the length of a channel ID codeword. Otherwise, it represents the length of a stop symbol.

C.1.4.4 Entropic Bandwidth

The average entropic BR for each channel for the GED encoding is given by:

$$\mathrm{BR}_{\mathrm{E-ga}} = -\frac{1}{n \times \mathrm{BP}} \sum_{a=1}^{n+S-2} (pg_a \times log_2(\widehat{pg_a})) \qquad \text{[bps/channel]} \quad \text{(C.25)}$$

C.1.5 GED encoding with delta-sampled channel IDs

A version of the GED encoding with delta-sampled channel IDs was not considered. This is because one would have to delta-sample the channel IDs, but would have to delta-sample the channel IDs within the same FR for the delta values to have any sense. E.g.

$$3\,1\,stop_2\,5\,1$$

would indicate that channels 3 and 4 had FR of 1, and that channels 5 and 6 had FRs of 2. Adding in variable length codewords for the combined relative probabilities of each delta-value and stop symbol seemed to be very overly complicated an encoding, frankly. It is theoretically possible, but a static pre-trained version seemed to be a significant amount of guesswork relative to the relative probabilities, and the hardware implementation was not attractive to the authors. As such, although perhaps feasible as an encoding, a GED encoding with delta-sampled channel IDs was not considered further by the authors.

C.1.6 Sample Histogram for Firing Rate mapping

The MUA histogram, for BP $\leq$ 100 ms, typically follows a decaying exponential. In other words, smaller FRs are more common than larger ones. However, this is not always the case. As such, automatically assigning shorter codewords to smaller FRs will not always give optimal compression. In Chapter 4, the use of a sample histogram was examined to address this problem. The beginning of each recording was used to fill a sample histogram. This histogram was then used to estimate the relative frequencies of the FRs for each channel. The most common FRs in the histogram were then, for the rest of the data in each channel, assigned the shortest codewords via sorting. This was referred to as mapping the most common FRs to the shortest codewords, given the sample histogram estimate. As such, some semi-adaptability was introduced into the SH encoders. This process is shown in Fig. 4.3.

Using a histogram for mapping the firing rate was extremely hardware efficient (for small S), since the sorting procedure was implemented using only combinatorial logic [59]. Furthermore, the sample histogram and sorting logic modules were shared across channels, with access to them multiplexed.

As such, for the SH and AH implementations of each encoding, we looked at the effect of including a sample histogram. We did not use them for the basic implementations, because there was no lossless compression and all codewords are in equal length. We considered histogram bin sizes of hs = 0, 2, 3, 4, 5, and 6 bits. In the case of 0 bits, no histogram, sorting or mapping was used. Otherwise, the beginning of each channel's recording was used to train the histogram. When 2^{hs} FR samples had been measured by the histogram for a channel j, the histogram training was ended for channel j. It was then sorted, and the mapping produced. The rest of the data was then compressed, after mapping. As such, only $v/\mathrm{BP} - 2^{hs}$ samples were compressed in the histogram-included versions of these encodings, where v = 100 s is the total length of each channel's recording.

For the asynchronous encodings, the effect is very important. With the FR histogram and mapping, one is no longer sending out the information from a channel j if it has a FR > 0. Rather, one is sending out its information if it has a FR different from its most common FR, as estimated by the histogram. This can greatly improve the compression if a channel's most common FR is not 0.

Table C.2: Average Logic cell and power across different parameters of SH-w SH-EED, and SH-DED at different channel counts

	SH-w	SH-EED	SH-DED n = 10	SH-DED n = 100	SH-DED n = 1000	SH-DED n = 10000	SH-DED n = 30000
Logic cells	230	237	258	271	290	303	308
Power (uW)	0.96	1.04	1.06	1.07	1.11	1.23	1.76

C.2 All encodings and implementation BR results

The full BR results for the basic, SH and AH encodings are given here.

C.3 Hardware Results

The work in this section, involving hardware design, was carried out by Z.Z., who also authored this section.

There is basically no difference in hardware costs between the Basic and SH implementations of each encoding. As such, we will show only the SH hardware results.

C.3.1 The number of logic cells is similar across encodings

Table C.2 shows the hardware requirements for the SH windowed (SH-w), explicit event-driven (SH-EED) encodings. We also show the delta event-driven (SH-DED) encodings as a function of channel count (averaged across all other parameters). This is because the SH windowed and SH explicit asynchronous hardware requirements do not increase with channel count However, the number of logic cells for the SH-DED encoding does depend on channel count, thus why different channel count values are shown for the SH-DED encoding.

It can be observed that there is little difference in hardware requirements between the encodings even in the most extreme case of $n = 30000$ for the SH-DED encoding. Therefore, we can conclude that the choice of encoding is basically unaffected by the number of required logic gates, although we would have a slight preference towards the SH-w and SH-EED encodings.

C.3.2 The memory requirements vary by encoding

Table C.3 shows the trend of required resources for the SH-EED and SH-DED encodings as a function of n, S and Δ_{max}. Importantly, these trends are in addition to the required resources for the SH-w encoding. As such, the SH-EED and SH-DED encodings require more resources than the SH-w encoding, and the additional resources follow the trend shown in Table C.3.

We can observe that the required logic cells increase as a function of channel count n and S in the SH-DED encoding, whereas for the SH-EED encoding it only increases as a function of S. However, the SH-DED encoding's memory requirements scale far better as a function of channel count n than the SH-EED encoding, since one can set $\Delta_{max} << n$. As such, as far as memory is concerned, SH-DED scales better with channel count, assuming Δ_{max} is set to a value smaller than n.

As such, we can conclude that SH-w is the most hardware efficient option. The SH-EED and SH-DED encodings require additional logic cells and memory relative to the SH-w encoding. However, the requirement for additional logic cells is minor for the SH-EED and SH-DED encodings, especially considering these logic cells are shared via multiplexing across channels. Between the SH-EED and SH-DED encodings, SH-DED consumes significantly less memory if Δ_{max} is set to a value $\Delta_{max} << n$.

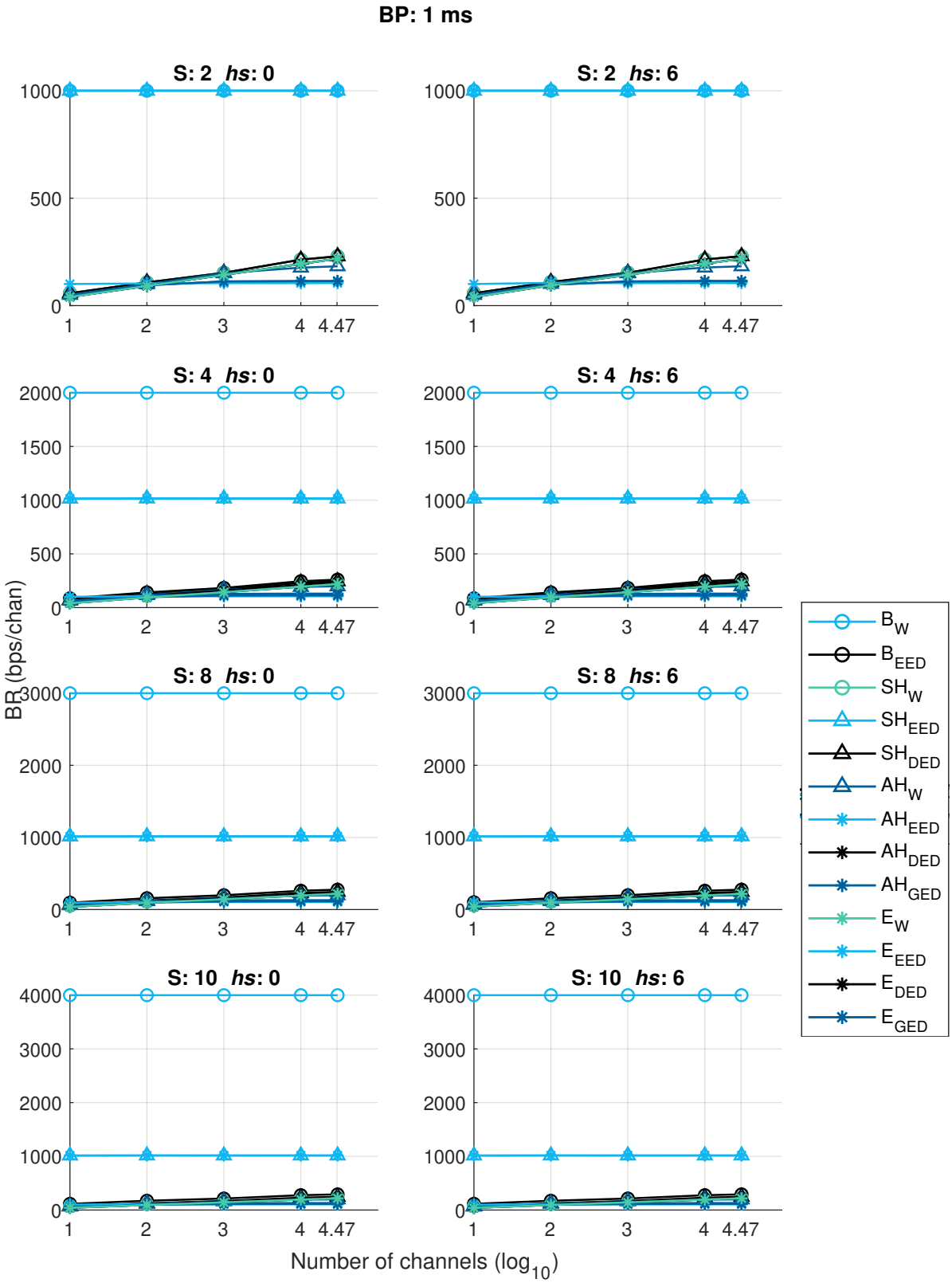


Figure C.1: BRs for communication schemes at a BTR of 1 ms.

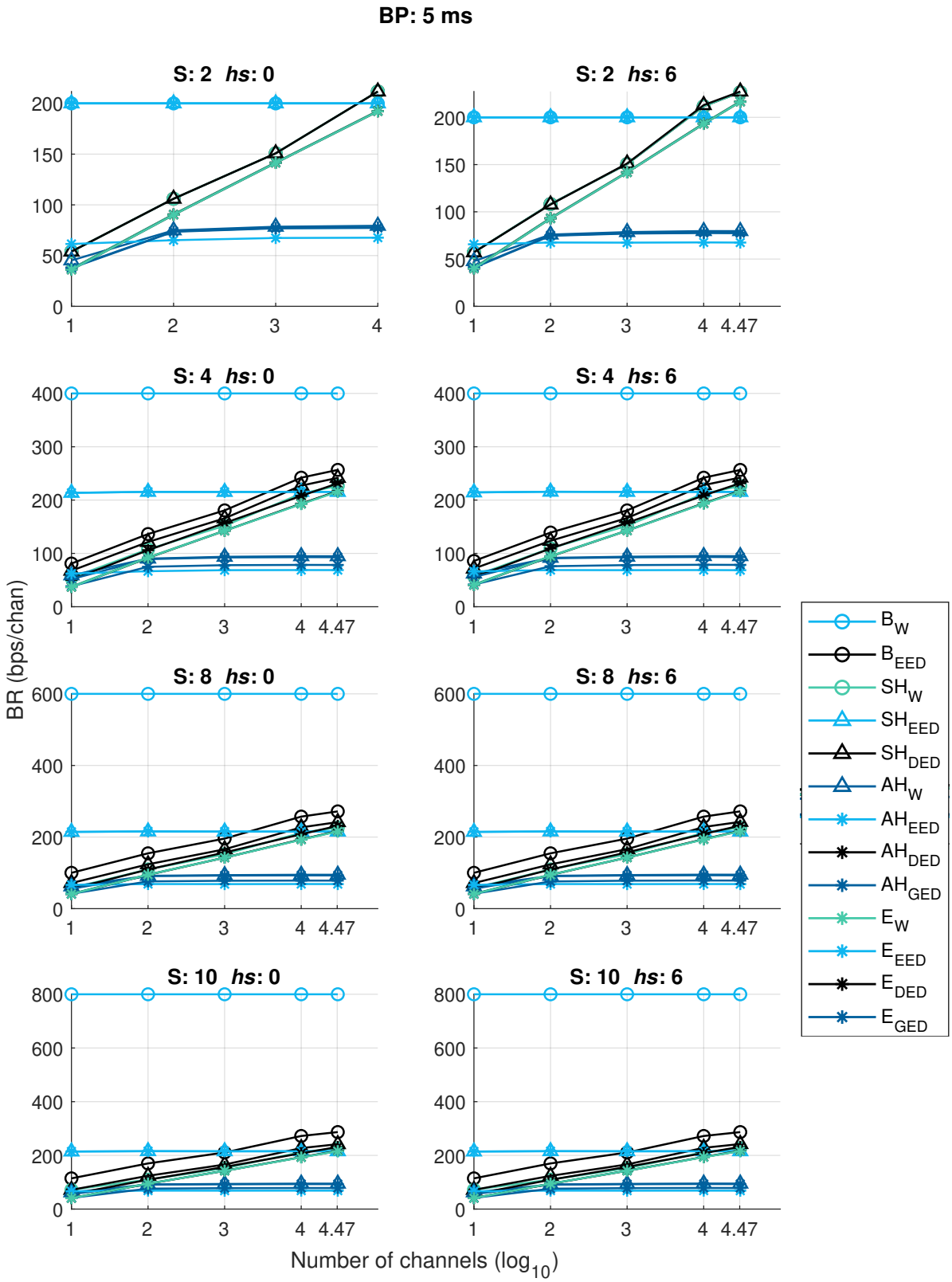


Figure C.2: BRs for communication schemes at a BTR of 5 ms.

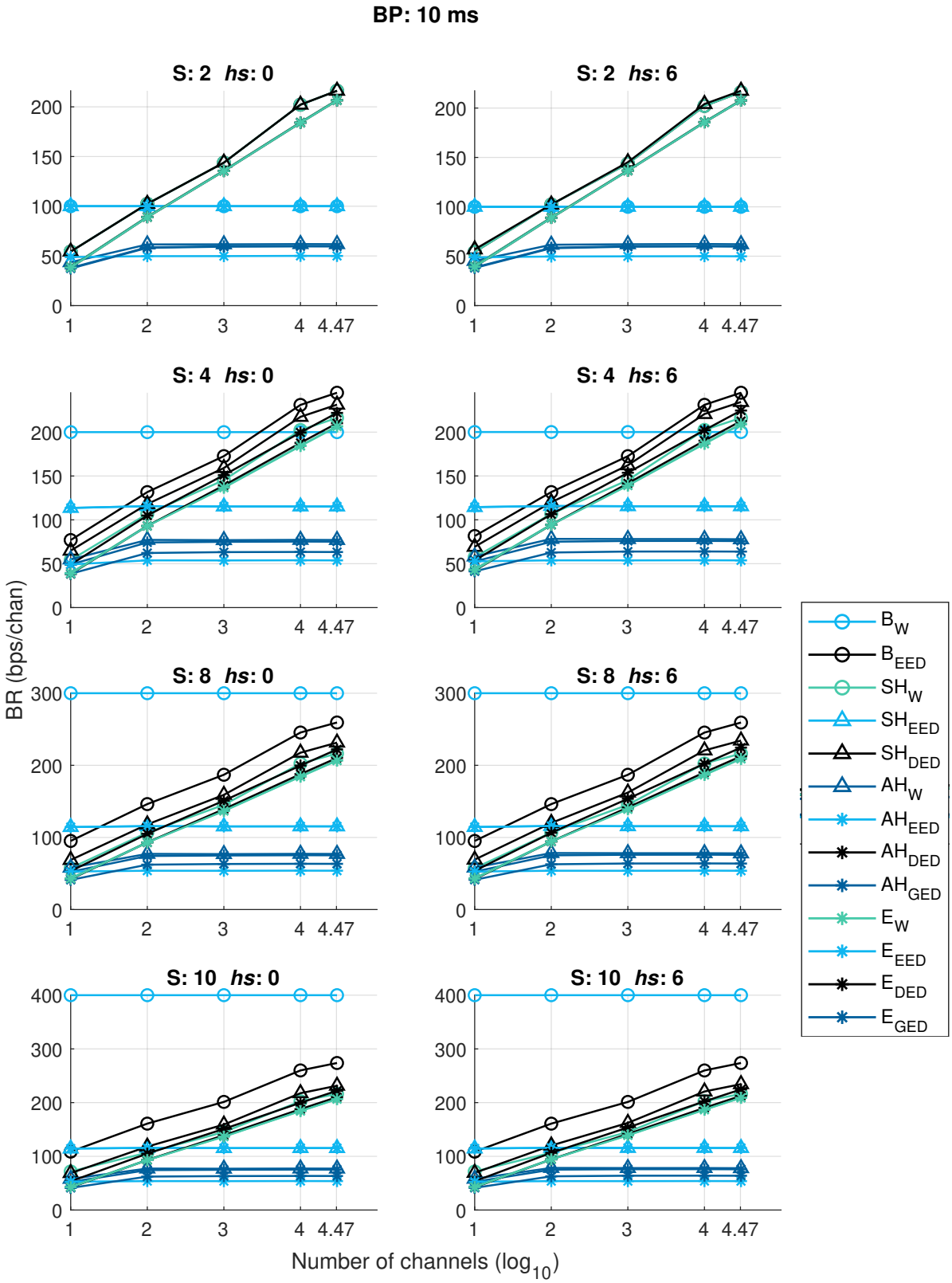


Figure C.3: BRs for communication schemes at a BTR of 10 ms.

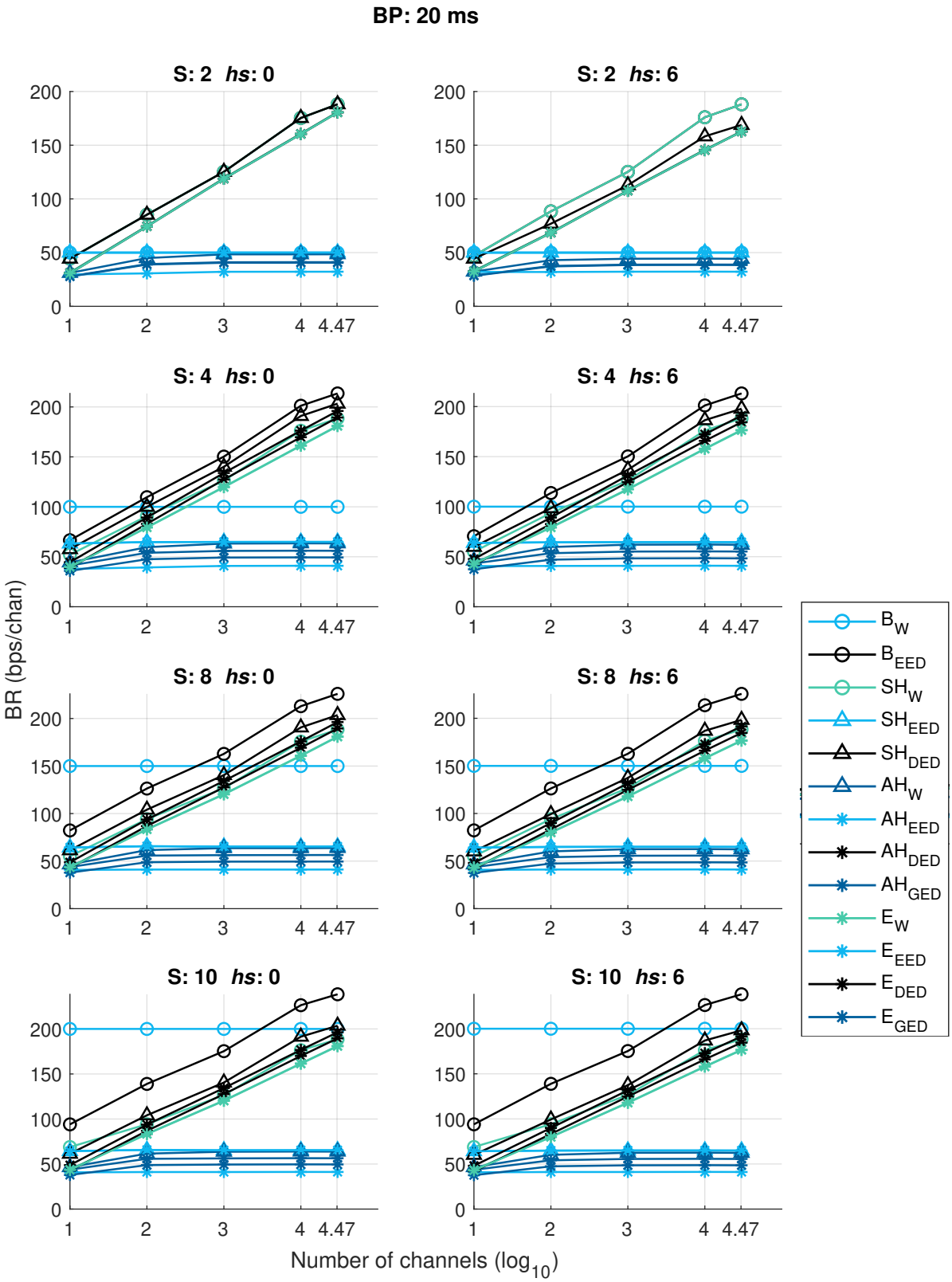


Figure C.4: BRs for communication schemes at a BTR of 20 ms.

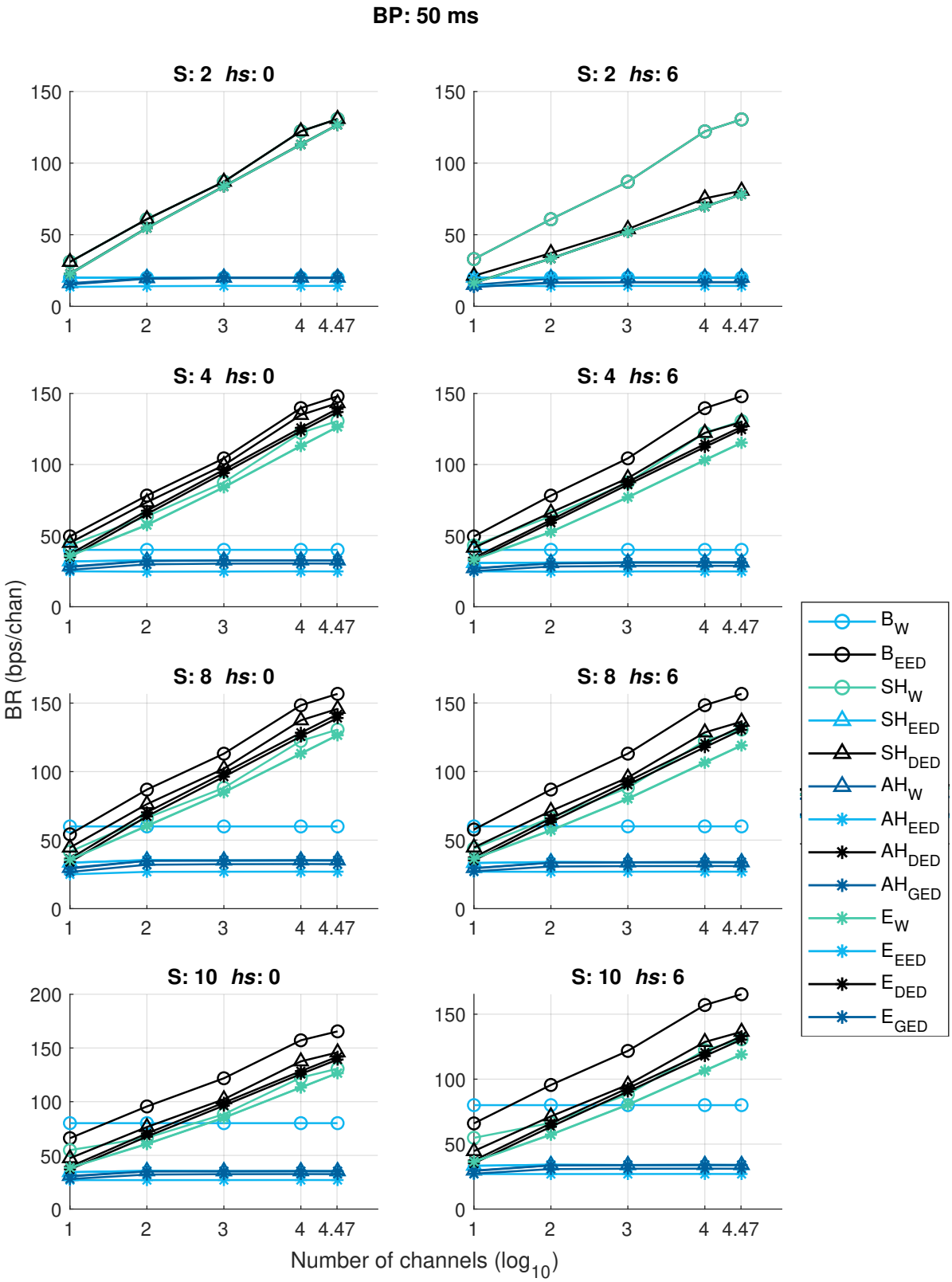


Figure C.5: BRs for communication schemes at a BTR of 50 ms.

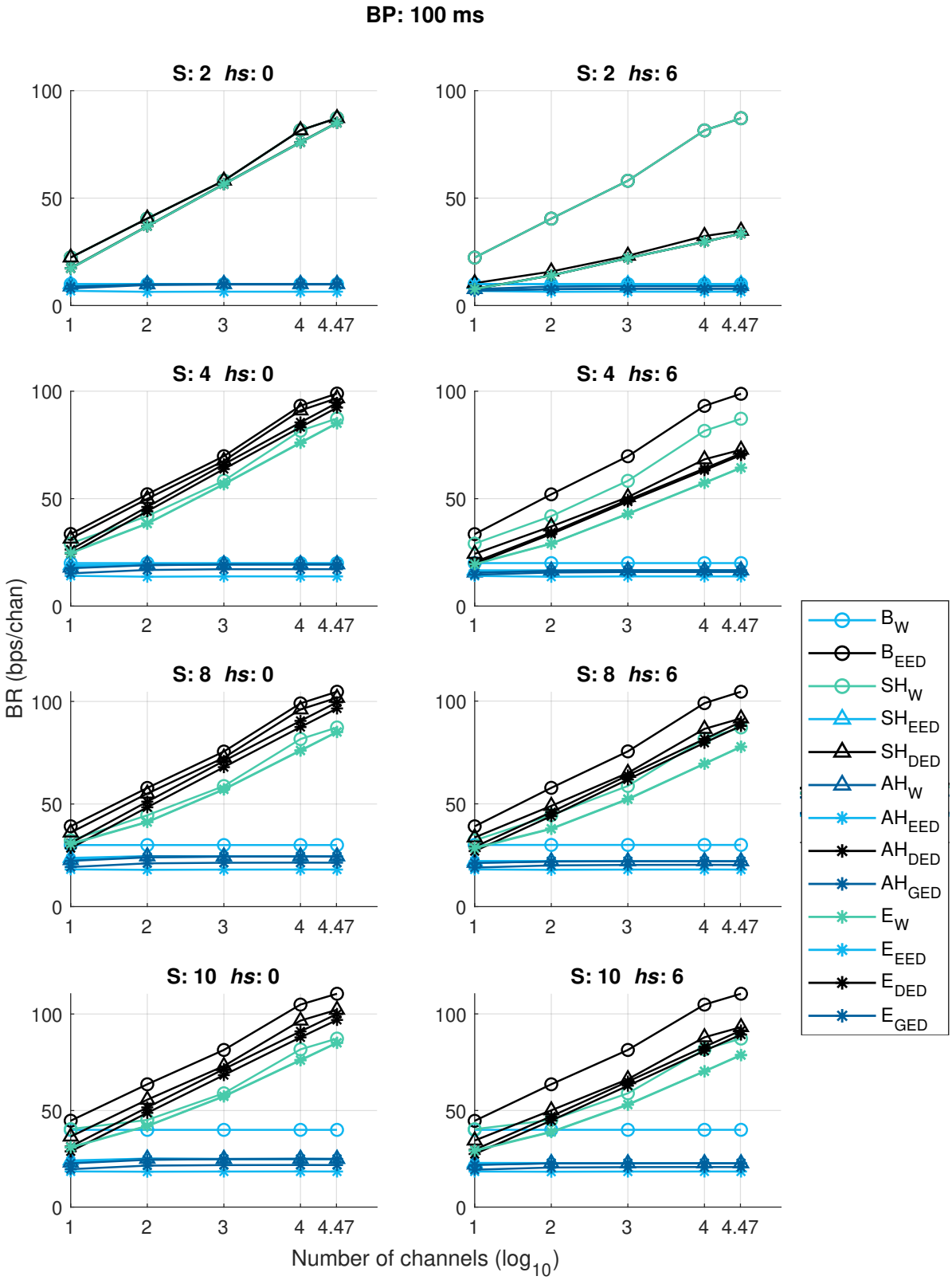


Figure C.6: BRs for communication schemes at a BTR of 100 ms.

Table C.3: Logic cell, RAM and Read Only Memory scale-up speed of SH-EED and SH-DED with different parameters. Importantly, these trends are relative to the SH-w encoding. As such, as they are positive values, they are in addition to that required for the SH-w encoding.

	SH-EED	SH-DED
Logic cells	log(S)	log(S)+log(n)
RAM	nlog(S)	nlog(S)
ROM	nlog(n)	Δ_{max}log(Δ_{max})

C.4 Hardware result analysis

The work in this section, involving hardware design, was carried out by Z.Z., who also authored this section.

C.4.1 Hardware setting selection

Hardware cost includes the area occupation and power consumption. These two factors determine the suitability of one algorithm for on-implant use. However, the ultimate solution for on-implant signal processing should use ASIC design to minimise the cost, while we assessed the hardware cost using FPGA. Assessing using FPGA can reduce the development time significantly, meanwhile FPGA power consumption is proportional to the ASIC design and its resource usage is directly related to the area occupation. Considering the enormous parameter space involves in this work, we used FPGA resource usage and core dynamic power as a reference to assess the suitability of different proposed algorithms and compare across them.

To break down the hardware cost, the resource usage consists of the logic cells and block RAMs. Logic cells consists of LUTs and flip-flops constructing the combinational and sequential processing logics. The block RAMs can be used to construct RAM and ROM for storing temporal logic status or Huffman codewords. The power consumption consists of the static power and dynamic power, while different implementation only contributes to the dynamic power. For the ease of presenting, we further breakdown the dynamic power consumption into the power contributed from the circuit working at clock frequency and the circuit working at bin frequency. The bin frequency circuit share is much lower than that of the clock frequency circuit. The mapper and encoder work at bin frequency and only consumes 0.06μW at 1 ms bin period. Longer bin period leads to the power consumption less than 0.01μW which is neglectable. Clock frequency power can be more significant and all power mentioned later will be the dynamic power per channel from the clock frequency circuit including binner, asyncing circuits, RAM and ROM.

Considering the superior performance of the SH over B implementation and the tiny cost of implementing the encoding (10% contribution to resources and neglectable effect on power consumption) comparing to AH, we prefer the SH architecture in this work. The GED implementation consumes much more than others does (over 800 logic cells). Therefore, the best architecture should be selected among SH-w, SH-EED and SH-DED.

Both SH-EED and SH-DED are built upon the SH-w implementation. The average logic cell usage and power consumption is given in Table.C.2, SH-EED only takes less than 5% extra logic cells which is neglectable. SH-DED uses more logic cells especially when channel count increases, but it is still acceptable if we consider the average resource usage per channel. The power consumption shows similar tends as the logic cells (Note that the power rocketing from $n = 10000$ to $n = 30000$ is from the addition memory usage). Therefore, when SH-EED or SH-DED can provide better compression performance, they are preferred instead of SH-w because of the tiny cost.

In order to make selection between SH-EED and SH-DED, Table. C.3 summarises how logic cells, RAM and ROM scales with different parameters. Details on how this table been derived is given in later two sections. One can notice that when the channel count increases, the logic cells of SH-DED increases in log-scale. However, the ROM usage of SH-EED increases in nlog-scale, which is much faster than the former one which grows with the Δ_{max}. We therefore prefer SH-DED especially when channel count is extreme. Setting a small Δ_{max} value can effectively limits the aggressive growth of the circuit size.

Next comes to select a reasonable Δ_{max} value. However, there is no clear clue on how the number of extra logic cells and power consumption are varying with Δ_{max} the power consumption is overall consistent. Therefore, one should trade off their ROM/area availably and BR requirement when selecting Δ_{max}. As the growing Δ_{max} has opposite effect on the ROM usage and BR reduction. Fig.C.7 shows the trend of power, bit rate and memory occupation (RAM+ROM) with different Δ_{max} values at bin period 1ms. 64, 128 and 256 are three sweet points to set Δ_{max}. Around 30% and 45% bit rate reduction can be achieved respectively with acceptable memory/size cost. Power consumption in this case is not a concern as it stays at similar levels among settings. When bin period exceed 1ms, there is no bit rate gain when Δ_{max} is greater than 64. Therefore, SH-DED with $\Delta_{max} = 64$ is recommended when async-compression can outperform the sync-compression.

It was decided to fix Δ_{max} at a value for each BP and n combination, which are the values that are relevant to Δ_{max}. This was done by comparing the hardware results to the compression results for different Δ_{max}. For each BP, the BR, total processing power and required memory for each n and Δ_{max} were observed. A design choice was then made, to find a Δ_{max} value that minimises BR, processing power and memory. An example is shown in Fig. C.7, for BP = 1 ms. As such, for a BP of 1 ms, we decided to set:

$$\Delta_{max} = \begin{cases} n, & \text{if } n \leq 100 \\ 64, & \text{if } n > 100 \end{cases} \tag{C.26}$$

For BPs larger or equal to 5 ms, it was found that there was little benefit to BR to increasing Δ_{max} beyond 64. As such, for BPs $\geq$ 5 ms, the following Δ_{max} values were set:

$$\Delta_{max} = \begin{cases} n, & \text{if } n \leq 10 \\ 64, & \text{if } n > 10 \end{cases} \tag{C.27}$$

This gave small values for Δ_{max}, minimising the required memory as given in Table C.3, while also giving good compression performance.

The resource usage of SH-w has been introduced in detail in [59], so here we here will only introduce the extra cost of SH-EED and SH-DED adding up to the SH-w later. As a reference, at bin period of 1ms, S = 3, histogram size = 2, SH-w occupies 129 logic cells and consumes $0.96\mu W$. The idea is to investigate how the extra hardware cost scales with certain parameters, and whether the extra cost in hardware worth the gain in BR reduction in certain BP.

C.4.1.1 SH-EED

The extra logic cell cost of SH-EED scales as $LC_{SH-EED} \propto log(S)$. In other words, in proportion to the bit width for S. However, comparing to the logic cell usage of the example SH-w implementation, this below 10 additional logic cell usage is neglectable.

When it comes to the memory usage, the RAM usage stays the same while it needs extra space of ROM to store the channel ID codewords. That ROM usage scales as $ROM_{SH-EED} \propto n \times log(n)$, as the bit width of the codeword scales logarithmically with the channel number.

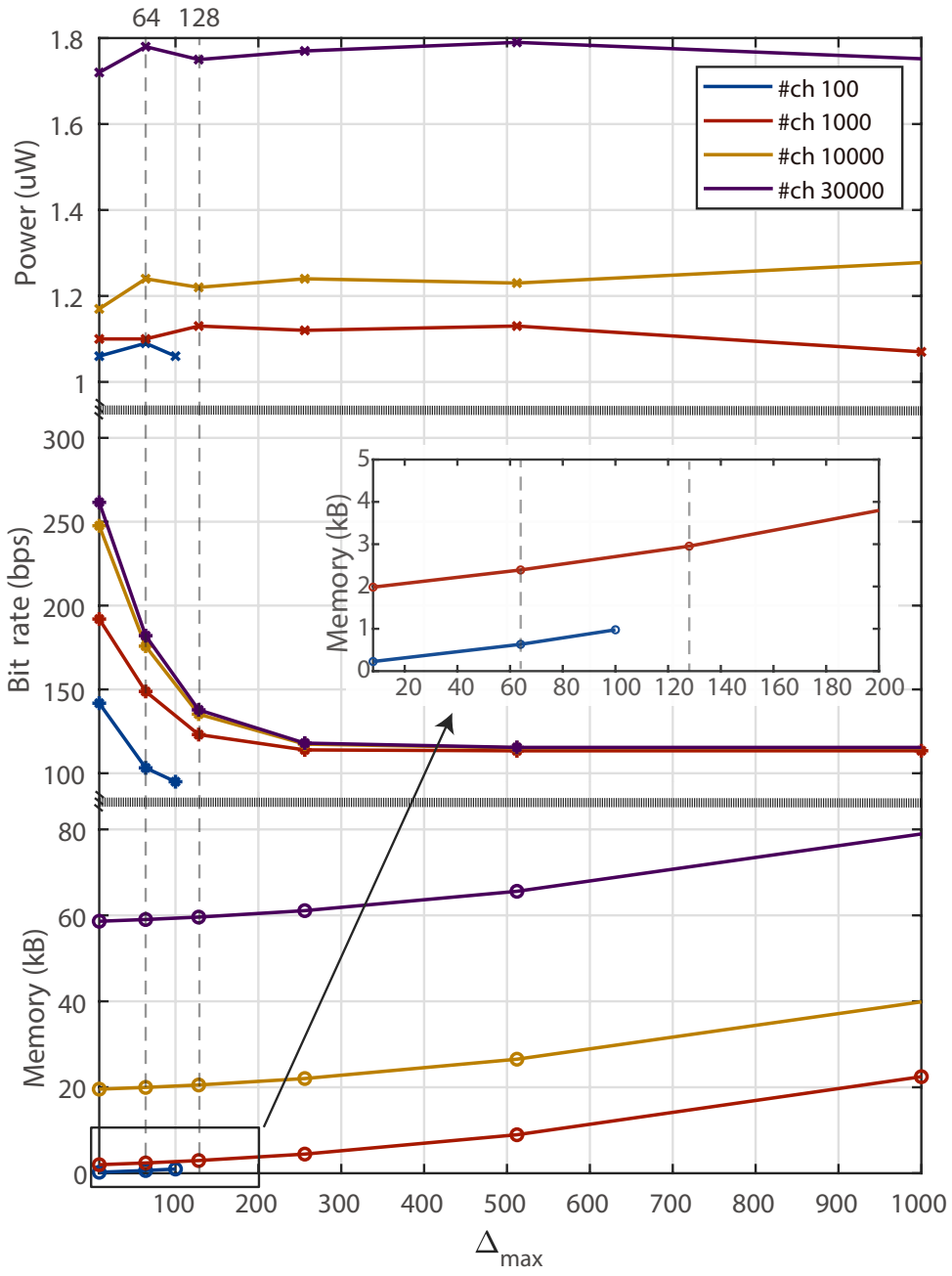


Figure C.7: The processing power consumption, bit rate and memory usage of the FPGA implementation with different Δ_{max} values with BP = 1 ms. We can see that processing power consumption is largely unaffected by Δ_{max}, and that channel count n is the main consideration for processing power. Δ_{max} mainly affects BR and memory consumption, where memory increases roughly linearly with Δ_{max} and BR decreases as a decaying exponential, with diminishing reductions in BR to increasing Δ_{max}. It can be observed that, depending on n, Δ_{max} = 64, 128, 256 are three good selections with balanced trade-offs.

As for the power consumption, our results suggest that the upscaling of the memory usage has less impact on power consumption, while the increasing logic cell usage does. Therefore, as the extra logic cells are neglectable, even though the memory occupation scales up quickly with the increasing channel count, the power consumption only increased for only 0.03μW from 10 channels to 1000 channels.

C.4.1.2 SH-DED

This architecture was built upon the SH-EED adding the logic counting the channel difference and the number of resetting. This extra logic is no longer neglectable. When there is no limitation on the max channel difference, i.e. $\delta_{max} = n$, the extra logic cells scale as $LC_{SH-EED} \propto log(n)$. The ROM occupation is the same as SH-EED case. As for the power consumption, we can only test the channel number under 1000 limited by the available onboard block RAMs. We have observed a more-than-linear growing extra power trend with increasing channel counts, comparing to the SH-DED. It needs less than 1% extra power at 10 channels, while that becomes 10% at 1000 channels.

In the case of limited max channel difference, tiny number of extra logic cells (No more than 10 logic cells) is need comparing to the $\Delta_{max} = n$ case. Its power upscaling also shares similar trending. The usage of ROM however can be highly constrained, which is translated to the reduced area occupation in future ASIC design. The power consumption is also reduced because of the reduced ROM usage. However, there is no clear clue on how the number of extra logic cells and power consumption are varying with Δ_{max} the power consumption is overall consistent. Therefore, one should trade off their ROM/area availably and BR requirement when selecting δ_{max}. As the growing Δ_{max} has opposite effect on the ROM usage and BR reduction.

Appendix D

Supplemental Material - Chapter 6: Minimum Requirements for the Processing and Compression of ESA

D.1 Training and testing recordings

The training and testing recordings are specified below in Table D.1.

D.2 Selected parameters for ESA data

The selected parameters for each BP from Fig. 6.4 are given in Table D.2. The equivalent results from Fig. 6.5 are given in Table D.3.

Table D.1: Training and Testing recordings from Sabes Raw Broadband data

Training Filenames	**Testing Filenames**
indy_20160624_03	indy_20160916_01
indy_20160915_01	indy_20160930_02
indy_20160921_01	indy_20161011_03
indy_20160927_04	indy_20161014_04
indy_20160927_06	indy_20161025_04
indy_20160930_05	indy_20161026_03
indy_20161005_06	indy_20161027_03
indy_20161006_02	indy_20161207_02
indy_20161007_02	indy_20161212_02
indy_20161013_03	indy_20170123_02
indy_20161017_02	indy_20170124_01
indy_20161024_03	indy_20170131_02
indy_20161206_02	
indy_20161220_02	
indy_20170127_03	

Table D.2: Hand-selected results from training data, chosen for high BDP and low BR, for each BP.

Train data results

f (Hz)	b (bits)	h	l (bits)	w (samples)	m (bits)	s (bits)	BP	BDP	BR
9000	12	3	5	128	5	2	14.22	0.724	96.24
8000	11	3	6	128	7	1	16.00	0.736	84.37
7000	11	3	6	128	6	1	18.29	0.699	68.60
7000	12	3	4	128	4	2	18.29	0.748	76.59
6000	12	3	4	128	4	3	21.33	0.766	66.33
5000	11	3	6	128	7	4	25.60	0.780	55.36
9000	12	3	4	256	4	1	28.44	0.783	46.16
9000	11	3	4	256	5	3	28.44	0.802	49.69
8000	12	3	6	256	6	1	32.00	0.787	42.04
8000	11	3	5	256	5	1	32.00	0.749	37.68
7000	10	3	4	256	5	1	36.57	0.731	33.96
7000	12	3	4	256	5	4	36.57	0.772	40.60
6000	12	3	4	256	4	1	42.67	0.784	31.61
6000	12	3	4	256	5	1	42.67	0.840	39.97
5000	12	3	6	256	6	3	51.20	0.814	26.11
9000	12	3	4	512	5	4	56.89	0.824	28.82
8000	12	3	3	512	5	3	64.00	0.829	26.53
7000	12	3	4	512	5	4	73.14	0.815	21.07
6000	12	3	5	512	6	5	85.33	0.794	19.62
6000	12	3	5	512	7	3	85.33	0.869	27.09
5000	11	3	3	512	4	3	102.40	0.729	13.34
5000	12	3	4	512	5	3	102.40	0.776	15.62
5000	12	3	6	512	7	3	102.40	0.795	18.88
9000	12	3	5	1024	7	2	113.78	0.826	14.93
9000	12	3	5	1024	7	3	113.78	0.846	19.01
8000	12	3	5	1024	6	2	128.00	0.815	11.82
8000	11	3	3	1024	6	3	128.00	0.834	15.91
7000	11	3	6	1024	7	3	146.29	0.768	9.04
7000	12	3	4	1024	6	5	146.29	0.815	13.88
6000	11	3	4	1024	6	5	170.67	0.768	8.98
6000	12	3	3	1024	5	5	170.67	0.784	10.68
6000	12	3	5	1024	7	5	170.67	0.814	12.80
8000	10	3	6	512	7	3	64.00	0.692	17.48
8000	10	3	5	512	7	3	64.00	0.724	18.51
8000	12	3	3	512	4	3	64.00	0.779	21.10
7000	11	3	3	512	4	4	73.14	0.742	18.38
7000	12	3	4	512	6	4	73.14	0.837	32.37

Table D.3: Test data results from the testing data, for same parameters as in Table D.2.

Test data results

f (Hz)	b (bits)	h	l (bits)	w (samples)	m (bits)	s (bits)	BP	BDP	BR
5000	11	3	3	512	4	10	102.40	0.64	13.02
5000	11	3	6	128	7	15	25.60	0.66	69.21
5000	12	3	4	512	5	10	102.40	0.71	16.60
5000	12	3	6	256	6	15	51.20	0.70	29.01
5000	12	3	6	512	7	10	102.40	0.73	18.88
6000	11	3	4	1024	6	5	170.67	0.70	9.46
6000	12	3	3	1024	5	5	170.67	0.70	11.39
6000	12	3	4	128	4	15	21.33	0.63	70.42
6000	12	3	4	256	4	15	42.67	0.68	32.72
6000	12	3	4	256	5	15	42.67	0.73	42.68
6000	12	3	5	512	6	10	85.33	0.74	20.23
6000	12	3	5	512	7	10	85.33	0.77	30.51
6000	12	3	5	1024	7	5	170.67	0.75	13.46
7000	10	3	4	256	5	15	36.57	0.58	34.43
7000	11	3	3	512	4	15	73.14	0.67	18.64
7000	11	3	6	128	6	15	18.29	0.56	72.53
7000	11	3	6	1024	7	5	146.29	0.65	10.01
7000	12	3	4	128	4	15	18.29	0.63	80.65
7000	12	3	4	256	5	15	36.57	0.73	49.69
7000	12	3	4	512	5	15	73.14	0.74	22.79
7000	12	3	4	512	6	15	73.14	0.77	32.94
7000	12	3	4	1024	6	5	146.29	0.75	15.06
8000	10	3	5	512	7	15	64.00	0.69	24.46
8000	10	3	6	512	7	15	64.00	0.58	19.19
8000	11	3	3	1024	6	10	128.00	0.75	16.96
8000	11	3	5	256	5	15	32.00	0.60	40.43
8000	11	3	6	128	7	15	16.00	0.63	110.07
8000	12	3	3	512	4	15	64.00	0.70	23.67
8000	12	3	3	512	5	15	64.00	0.75	33.26
8000	12	3	5	1024	6	10	128.00	0.71	12.46
8000	12	3	6	256	6	15	32.00	0.69	50.29
9000	11	3	4	256	5	15	28.44	0.67	52.13
9000	12	3	4	256	4	15	28.44	0.66	49.36
9000	12	3	4	512	5	15	56.89	0.74	28.95
9000	12	3	5	128	5	15	14.22	0.62	116.97
9000	12	3	5	1024	7	10	113.78	0.78	20.80

Appendix E

Published Code and Results

E.1 Chapter 4 - Static Huffman Compression of MUA

The analysis Python code and FPGA Verilog code and designs for the windowed MUA compression have all been made publicly available at [103]. The formatted data and results have been made available at [104].

E.2 Chapter 5 - Comparison to Event-Driven Architectures for Compressing MUA

All code and results for the event-driven MUA compression are publicly available at [107].

Appendix F

List of Publications

Towards a Distributed, Chronically-Implantable Neural Interface Nur Ahmadi, Matthew L Cavuto, Peilong Feng, Lieuwe B Leene, Michal Maslik, Federico Mazza, Oscar Savolainen, Katarzyna M Szostak, Christos-Savvas Bouganis, Jinendra Ekanayake, Andrew Jackson, Timothy G Constandinou. 2019 9th International IEEE/EMBS Conference on Neural Engineering (NER), 719-724, 2019

Lossless Compression of Intracortical Extracellular Neural Recordings Using Non-Adaptive Huffman Encoding OW Savolainen, TG Constandinou 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), 4318-4321, 2020

Investigating the Effects of Macaque Primary Motor Cortex Multi-Unit Activity Binning Period on Behavioural Decoding Performance OW Savolainen, TG Constandinou 2021 10th International IEEE/EMBS Conference on Neural Engineering (NER), 436-439, 2021

Predicting Single-Unit Activity from Local Field Potentials With LSTMs OW Savolainen, TG Constandinou. 2020 42nd Annual International Conference of the IEEE Engineering in Medicine & Biology Society (EMBC), 884-887, 2020

Algorithm and Hardware Considerations for Real-Time Neural Signal On-Implant Processing Zheng Zhang, Oscar W Savolainen, and Timothy G Constandinou. Journal of Neural Engineering, 19(1):016029, 2022

Hardware-Efficient Compression of Neural Multi-Unit Activity OW Savolainen, Z Zhang, P Feng, TG Constandinou. IEEE Access, 2022

Ultra Low Power, Event-Driven Data Compression of Multi-Unit Activity OW Savolainen, Z Zhang, TG Constandinou. bioRxiv, 2022

Development of an Ultra Low-Cost SSVEP-Based BCI Device for Real-Time On-Device Decoding James Teversham, Steven S Wong, Bryan Hsieh, Adrien Rapeaux, Francesca Troiani, Oscar Savolainen, Zheng Zhang, Michal Maslik, Timothy G Constandinou. bioRxiv, 2022

The Significance of Neural Inter-Frequency Power Correlations OW Savolainen. Scientific reports 11 (1), 1-23, 2021